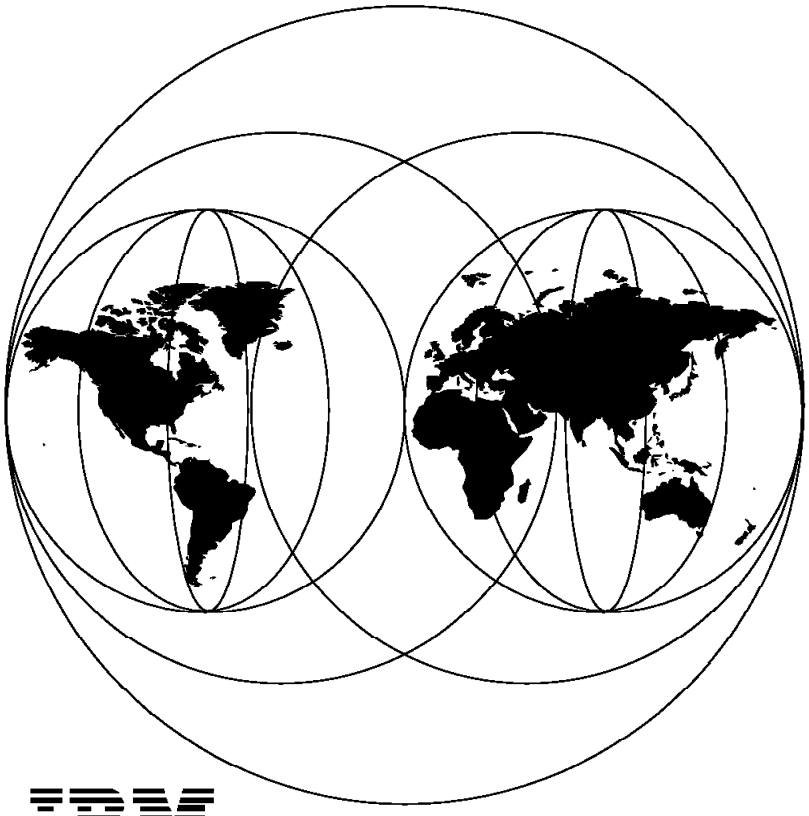
SG24-4721-00

Accessing OS/390 OpenEdition MVS from the Internet

July 1996



IBM

International Technical Support Organization
Raleigh Center

International Technical Support Organization

SG24-4721-00

Accessing OS/390 OpenEdition MVS from the Internet

July 1996

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix F, “Special Notices” on page 243.

First Edition (July 1996)

This edition applies to OpenEdition MVS Version 5 Release 2 Modification Level 2 of MVS/ESA System Product (program number 5655-068 and 5655-069), TCP/IP for MVS Version 3 Release 1 including the OpenEdition Applications Feature (program number 5655-HAL), Advanced Communications Functions for Virtual Telecommunications Access Method (ACF/VTAM) VTAM Version 4 Release 3 for MVS/ESA including the AnyNet feature (program number 5695-117) and Version 1.0 of the Internet Connection Server for MVS/ESA (program number 5655-156).

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 678
P.O. Box 12195
Research Triangle Park, NC 27709-2195

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1996. All rights reserved.**
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

© Copyright IBM Corp. 1996

Preface

This redbook provides information about how to turn an OpenEdition system into a productive Internet host. It focuses on how to use both TCP/IP for MVS and AnyNet for MVS to connect an OpenEdition host to an Internet and it gives guidance on how to plan for, implement and integrate the TCP/IP for MVS OpenEdition Applications Feature into the OpenEdition environment.

This book was written for system programmers and network administrators who are planning to use TCP/IP V3R1 for MVS, the TCP/IP V3R1 for MVS OpenEdition Applications Feature, AnyNet for MVS and the Internet Connection Server for MVS/ESA in an OpenEdition environment. Some basic knowledge of these products is assumed.

How This Redbook Is Organized

This redbook is organized as follows:

- Chapter 1, “OpenEdition, TCP/IP and AnyNet Overview”

  This chapter introduces the OpenEdition environment.

- Chapter 2, “Security”

  This chapter discusses how to establish RACF security setup for socket applications in the OpenEdition environment.

- Chapter 3, “Setting Up the OE Resolver”

  This chapter guides you through the tasks that are associated with configuring the OpenEdition resolver function.

- Chapter 4, “Setting Up the AF_INET Transport Providers”

  This chapter gives you detailed instructions for setting up TCP/IP for MVS and AnyNet MVS as AF_INET transport providers for socket applications that execute in the OpenEdition envrionment.

- Chapter 5, “TCP/IP for MVS OpenEdition Applications Feature”

  This chapter describes how to install, customize and test the TCP/IP for MVS OpenEdition Applications Feature.

- Chapter 6, “Web Services”

  This chapter describes how you can use the OE FTPD server and TelnetD server from web browsers.

- Chapter 7, “Tracing OpenEdition Socket Applications”

  This chapter gives you detailed information on how you can perform various traces in case your OpenEdition socket programs do not behave as you expected them to behave.

- Chapter 8, “Putting Things Together”

  This chapter is full of small useful hints and tips.

© Copyright IBM Corp. 1996

The Team That Wrote This Redbook

This redbook was produced by a team of specialists from around the world working at the Systems Management and Networking ITSO Center, Raleigh.

Alfred B. Christensen is a senior ITSO specialist at the Systems Management and Networking ITSO Center, Raleigh. Before joining the ITSO two years ago, Alfred B. Christensen worked in IBM Denmark as a senior systems engineer.

Hans-Dieter Mertiens is a senior technical marketing support specialist in IBM Germany.

Konni Losert is an account systems engineer in IBM Germany.

Seiko Suzuki is an information technology specialist in IBM Japan.

Thanks to the following people for their invaluable contributions to this project:

Kathryn Casamento
Gail Wojton
Carla Sadtler
Systems Management and Networking ITSO Center, Raleigh

Greg Ames
Jim Donoho
Janet Wolf
AnyNet MVS Development, IBM Research Triangle Park

Ginny Chung
Mary K. Deuser
TCP/IP for MVS Development, IBM Research Triangle Park

Larry Benton
Language Environment Development, IBM Kingston

Comments Welcome

We want our redbooks to be as helpful as possible. Should you have any comments about this or other redbooks, please send us a note at the following address:

redbook@vnet.ibm.com

Your comments are important to us!

Chapter 1. OpenEdition, TCP/IP and AnyNet Overview

The first chapter of this book is an introduction to the various topics and components that you will have to work with in order to install, customize and use the TCP/IP for MVS OpenEdition Applications Feature and the Internet Connection Server for MVS.

The tasks that you will have to both plan for and perform to make these functions available to your users require that you, or the group of people to which you belong, posess the combined skills of a UNIX specialist, an MVS specialist and a networking specialist.

If your background is in traditional MVS programming or systems programming, the OpenEdition terminology may at first seem to be somewhat confusing. If your background is in the UNIX environment, the opposite is equally true: the terminology used in the traditional MVS environment may seem confusing to you.

The objective of the first chapter of this book is to acquaint you with all the areas of expertise that are required to turn your OpenEdition environment into a productive open systems host that allows users in your TCP/IP and SNA network to gain access via standard TCP/IP client applications, such as telnet, FTP, REXEC, RSH, or Web browsers.

This chapter includes the following topics:

- 1.1, Software Component Overview
- 1.2, Introduction to OpenEdition
- 1.3, OpenEdition Processes and Threads
- 1.4, OpenEdition User Identification
- 1.5, UID/GID Assignment to a Process
- 1.6, Sockets in the OpenEdition Environment
- 1.7, OpenEdition Resolver Configuration Data
- 1.8, Server Programs in OpenEdition
- 1.9, Environment Variables
- 1.10, AnyNet and OpenEdition

© Copyright IBM Corp. 1996

1.1 Software Component Overview

This book covers a spectrum of software products and features. Table 1 provides a short overview of when you would need which product or feature.

Table 1. When You Need Which Product or Feature

What you want to do	Products or features needed
You want to access AF_INET socket applications that run in the OpenEdition environment.	You need an AF_INET transport provider, which can be one or both of the following: 1. TCP/IP for MVS 2. AnyNet MVS configured as a Sockets over SNA access node
You want to access Hierarchical File System files from remote FTP clients, access OpenEdition shell functions from non-3270 workstations in line-mode or in raw-mode, remotely execute OpenEdition shell commands, or develop and run ONC/RPC or X-Windows applications in the OpenEdition environment.	You need the TCP/IP for MVS OpenEdition Applications Feature[1] in addition to an AF_INET transport provider, which can be one or both of the following: 1. TCP/IP for MVS 2. AnyNet MVS configured as a Sockets over SNA access node
You want to use your MVS system as a Web server.	You need the Internet Connection Server for MVS in addition to an AF_INET transport provider, which can be one or both of the following: 1. TCP/IP for MVS 2. AnyNet MVS configured as a Sockets over SNA access node

Note:

1. The TCP/IP for MVS OpenEdition Applications Feature is a feature of the TCP/IP for MVS product, so you need to order TCP/IP for MVS with the base feature and the TCP/IP for MVS OpenEdition Applications Feature. When the TCP/IP for MVS OpenEdition Applications Feature has been installed, you may access the applications via both TCP/IP for MVS and AnyNet MVS.

1.2 Introduction to OpenEdition

Beginning with MVS/ESA Version 4.3 a new type of application program interface was added to the MVS platform. This new application program interface was mainly based on the definitions 1003.1 and 1003.2 of the IEEE working group 1003. This working group was formed to consolidate the various flavors of UNIX operating systems in use. The definitions that were made by this working group included both a *C* programming API and an interactive environment that is generally referred to as the *shell*. These definitions are very close to the operating system and are better known under the name POSIX, which stands for Portable Operating System Interfaces - UNIX.

Though aimed at unification, these definitions did not cover totally what one needs to implement portable C applications on various UNIX platforms. For example, the definitions did not include the popular BSD socket application program interface. Networking is part of a separate set of definitions but these are still not in an agreed-upon state.

Partly for that reason, groups other than IEEE began to study functions and services one needs to implement UNIX applications on various platforms. These groups made a more practical approach. First they took all definitions that had already been made by the POSIX working group. Secondly, they scanned various UNIX applications for the interfaces that were used by these applications. They ended up with 1170 definitions, which then gave name to this set of definitions: SPEC1170.

Yet another organization, X/Open, deals with definitions of standard interfaces. The definitions made by X/Open are also based on already existing standards or practices. X/Open′s definitions are published in the *X/Open Portability Guides* (XPG). One part of these guides define what the underlying operating system has to provide in terms of application program interfaces and services. For this part, X/Open integrated SPEC1170 into a version of the portability guides that is generally known as XPG Issue 4 Version 2 (or for short XPG 4.2).

It should be noted that none of the definitions require a specific hardware or software product. All these definitions describe *what* to implement, not *how* to implement. For that reason you will find these definitions implemented not only on operating systems with a UNIX heritage, but also on other operating systems, such as MVS/ESA.

In parallel to all this development, UNIX itself was also changing. After being owned by AT&T, UNIX Software Laboratories was sold to Novell in 1993. After being part of Novell for a short time, the term UNIX was sold to X/Open. In other words, the term UNIX is no longer bound to the operating system as when it was delivered by AT&T. Today it is a brand owned by X/Open. X/Open verifies the implementation of the UNIX functions as they are defined in the *X/Open Portability Guides* and certifies the specific implementation. Through this branding mechanism it is guaranteed that a branded system corresponds to the UNIX definitions that are described in XPG 4.2.

There are different levels of branding. MVS/ESA SP 5.2.2 received the base branding at the end of 1995. OS/390 is expected to have the full UNIX branding in 1996.

OpenEdition will be the implementation of UNIX as it is defined by X/Open in the *X/Open Portability Guide 4.2*. The implementation of OpenEdition is made in such a way that UNIX functions coexist with the traditional MVS functions. This means that an MVS/ESA SP 5.2.2 or OS/390 system with OpenEdition can be used to run the daily well-known MVS applications, as every MVS release before has done. In addition to that, it adds an almost complete UNIX interface to the set of services that are available to application programs.

1.2.1 ISO/ANSI-C, POSIX, and UNIX

Before discussing some further important new functions, let us clarify the relationship between ISO/ANSI-C, POSIX, and UNIX.

They all deal with the general-purpose C programming language. Using the C/MVS programming language, you can write portable code conforming to the following standards: ISO (ANSI), POSIX, or XPG4. What is different is the scope of the corresponding definitions. See Figure 1 for an overview of how these standards relate to each other.

Note: ANSI/ISO 9899:1990 (1992); American National Standards Institute, C language, was formerly covered by ANSI X3J11.159-1989 C.

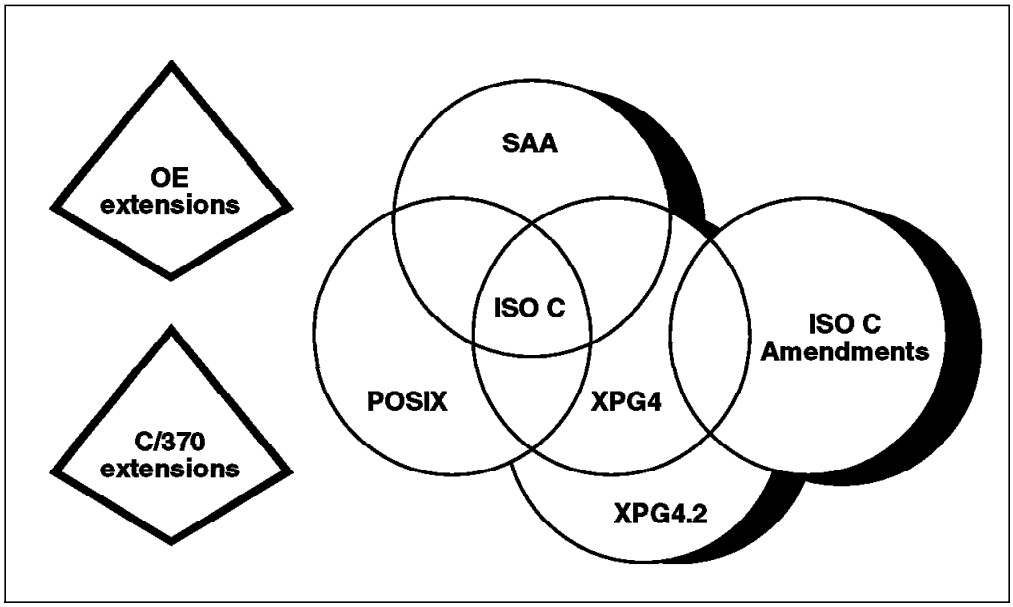


Figure 1. C Standards

As you can see, XPG 4.2 currently covers the widest area of C function calls and headers. As an example, the strchr() function is part of both ISO/ANSI C and XPG 4.2, whereas the getsockname() function is covered by XPG 4.2 only.

In general, C by itself lets you write highly portable code. As long as you do not use dirty tricks (see 8.4.1, “Pitfalls” on page 197) your code can be recompiled on the OpenEdition platform without major recoding or, even worse, redesign.

1.2.2 OpenEdition Components

OpenEdition provides two system interfaces on the MVS operating system: an application programming interface (API) and, optionally, a shell interface similar to the traditional UNIX Korn shell. Though not required, it is generally recommended that you install the Shell and Utilities component.

Note: Please note that if you plan to install the TCP/IP for MVS OpenEdition Applications Feature, the Shell and Utilities component is required.

The application programming interface can be used by MVS programs that execute in any of the environments that are supported by MVS, for example, in the new OpenEdition shell environment, or in the traditional MVS environments such as in MVS batch jobs, in TSO address spaces or in started tasks. Portable programs can be written in C or C++ and use facilities such as the following:

- POSIX threading services to create multi-threaded applications.
- Distributed Computing Environment (DCE) base services to create Distributed Computing Environment/Remote Procedure Call (DCE/RPC) server or client applications.
- Standard UNIX interprocess communication (IPC) facilities (for example, pipes and streams).
- BSD sockets to communicate with other programs on the same OpenEdition system (AF_UNIX sockets) or with programs on other systems (AF_INET sockets).

Programs that run on an MVS or OS/390 system with OpenEdition enabled can use traditional MVS services, OpenEdition services or a mix of traditional MVS services and OpenEdition services.

The optional Shell and Utilities component is an execution environment somewhat similar to a TSO/E environment. The interactive shell interface adds an interface to MVS that is familiar to UNIX users. The shell interface allows the user to execute shell commands, run application programs or execute shell scripts that consist of execution logic, shell commands and program invocation. A shell script is similar to a TSO command list or a REXX program.

The shell interface is available in interactive mode from line-mode or raw-mode terminals that connect to OpenEdition, from TSO/E in line-mode, and in batch mode where shell work can be submitted to a background environment as a normal MVS batch job using the BPXBATCH utility program.

For users that are familiar with the ISPF interface of TSO/E, an ISPF-based shell interface (ISHELL) is supplied that allows an ISPF user to perform many of the same functions that an interactive shell user is able to perform with the native shell interface.

OpenEdition support is provided by the following components:

- OE System Services, providing OE services to respond to requests from programs and the shell environment.
- OE Application Services, containing the code to support full-screen applications, remote login and automount support.
- OE Shell and Utilities feature, interpreting commands from interactive users or from shell scripts and requests MVS services in response to those commands.
- OE Debugger feature (dbx), helping application programmers to debug programs written in C.

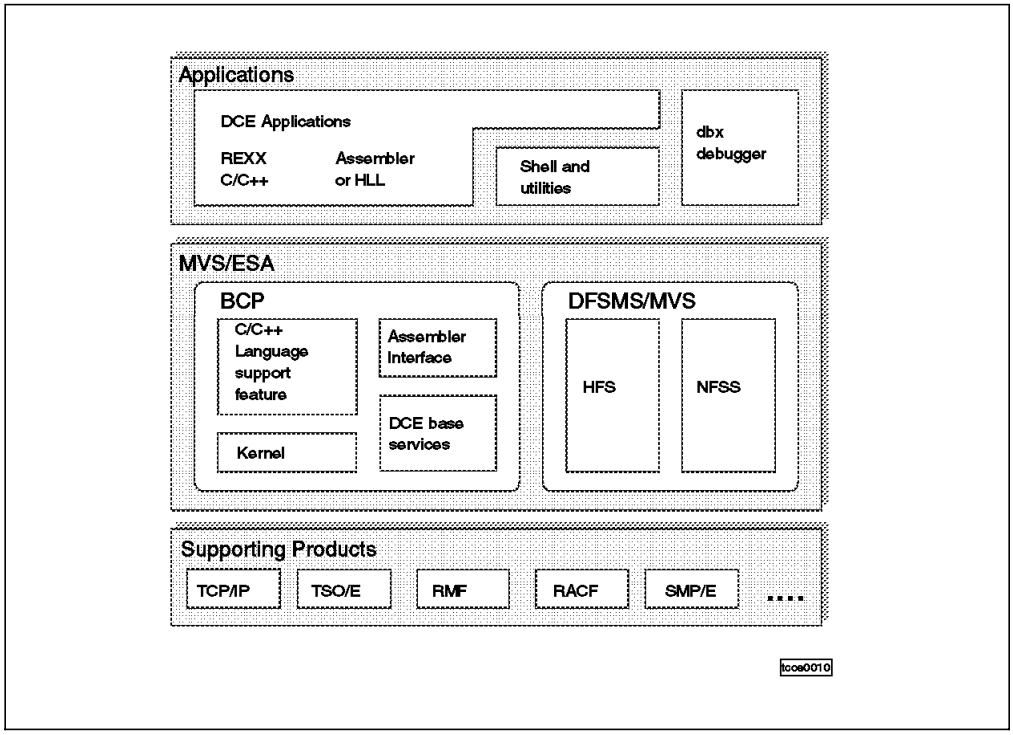


Figure 2. Product Components in an OpenEdition System

To implement the components mentioned above, the following traditional MVS products provide support as an integral part of their functions:

- DFSMS/MVS - implements a Hierarchical File System in the MVS environment.

  It must be pointed out that this file system is a completely new type of file system in the MVS environment. This file system implements the typical characteristics of a UNIX file system:

  – Files may have names up to 1023 characters in length.

  – File names are case sensitive.

  – File names may include special characters, such as spaces.

  – Files are byte oriented as opposed to record oriented.

  – Files are arranged in directories.

  The Hierarchical File System is implemented in “containers” that are normal MVS data sets. A Hierarchical File System container has an MVS data set name with a DSNTYPE of HFS. The container data set may be handled by traditional DFSMS/MVS storage management tools, such as DFHSM or DFDSS. You decide which parts of the total Hierarchical File System are implemented in which containers. You may even implement the full Hierarchical File System in one container, or you may spread parts of the file system hierarchy over a number of individual container data sets.

  DFHSM and DFDSS operates at the HFS container level, while ADSTAR Distributed Storage Manager (ADSM) supports backup at the individual file level if you have the ADSM OpenEdition client installed.

  The Hierarchical File System is one of the important pieces that allows an MVS system to have the look and feel of a UNIX system.

- RACF - implements support for defining OpenEdition users with the attributes that are required in a UNIX environment, for example, with a UNIX UID and a UNIX GID.
- SMP/E - implements support for installing and maintaining program product objects in the Hierarchical File System.
- TCP/IP for MVS and AnyNet MVS - implement support that allows OpenEdition socket applications to communicate with socket applications on remote TCP/IP hosts.

Before OpenEdition services can be used, the OE System Services component, the OE Kernel address space (the OMVS address space), must be installed and started. The OE Shell and Utilities feature and the OE Debugger are optional.

APPC/MVS is a prerequisite for OpenEdition services and must be active on the MVS system before the OE Kernel address space is started.

The Hierarchical File System container data sets must be allocated on SMS managed volumes, so you must also have SMS enabled on your MVS system in order to use the OpenEdition functions.

The redbook *OpenEdition MVS Installation and Customization Starter Kit*, SG24-4529, provides step-by-step instructions for installing, customizing and using the OpenEdition product set. It also includes a diskette containing machine-readable sample job control statements (JCL) and programs.

The C/C++ language support feature of OpenEdition defines all the data structures and functions that are used by C/C++ programs to access the OpenEdition services. Most of the services are also available via standard call interfaces that can be used by, for example, assembler programs that need the services of the OE Kernel address space. See Figure 3 for an overview of the relationships between the C/C++ functions and the callable services of OpenEdition.

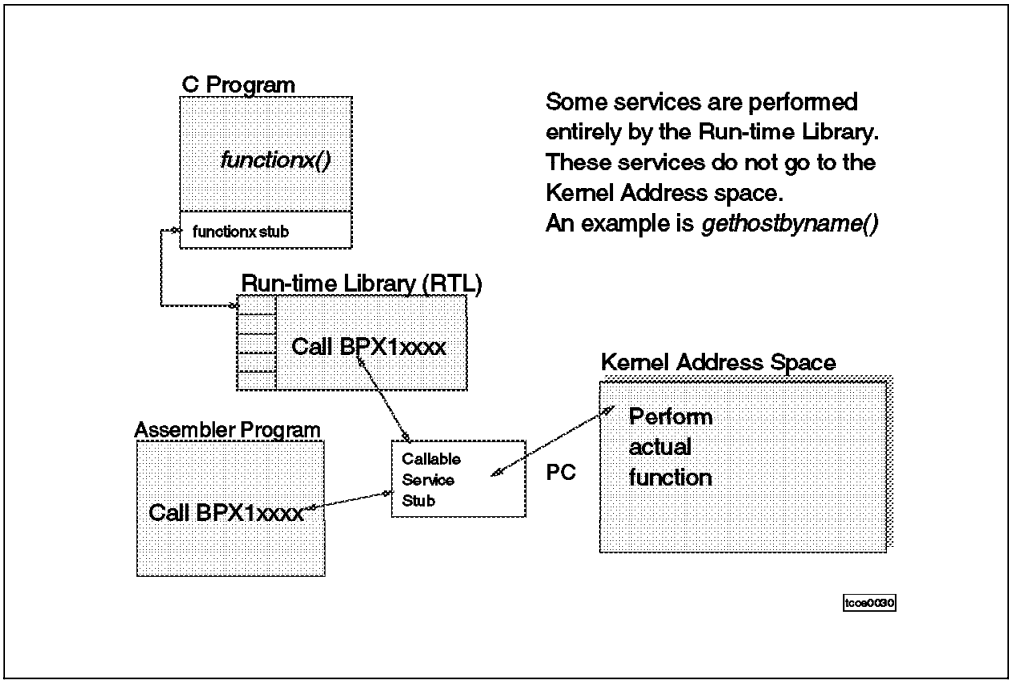


Figure 3. OpenEdition Callable Services

The OpenEdition callable services are documented in *Assembler Callable Services For OpenEdition MVS*, SC23-3020. Even if you are developing programs in C or C++, this book might be useful to you as a reference for diagnosing return codes and reason codes that are passed back to your application program. This is because most of the C function calls actually are passed on by the C language support to the OE Kernel address space via the assembler callable services. One example is the C function fork(), which is passed on as a call to the BPX1FORK assembler callable service.

All callable services are invoked with three standard 4-byte binary integer return parameters:

RV Return Value. This parameter may be used for different purposes depending on the specific call, but if the return value is -1, it means that the call failed. When the returned value is -1, the return code field holds a generic error number and the reason code field holds a specific reason code that explains in more detail what error situation a program encountered.

Note: Please note that if the return value field is *not* -1, the value of the return code and reason code fields are unpredictable.

RC Return Code. The parameter holds the generic error number (in general called the ERRNO field). These error numbers are defined in Appendix A of *Assembler Callable Services For OpenEdition MVS*, SC23-3020.

RS Reason Code. The parameter holds the specific reason code (sometimes referred to as ERRNO-junior). These reason codes are defined in Appendix D of *Assembler Callable Services For OpenEdition MVS*, SC23-3020. The first two bytes are reason code qualifiers and the remaining two bytes hold the actual reason code.

Even if you do not develop your own OpenEdition application programs, you may still need to diagnose error messages that are issued by standard components of the OpenEdition system. Most error messages that are related to unexpected return values from a call to the OE Kernel address space include these three parameters. The following is an example of an error message that may be issued by TCP/IP for MVS when the TCP/IP system address space tries to connect to the OE Kernel address space:

```
EZY2141E OpenEdition-TCP/IP Connection error for TCPUSER
         -BPX1SOC, FFFFFFFF,0000009C,0B0C00FB
```

BPX1SOC This is the name of the OpenEdition callable service that was used by TCP/IP for MVS when an error occurred. This call is a socket() call to open a new socket descriptor in the OE environment.

FFFFFFFF This is the return value parameter. It has a value of -1, which indicates an error condition.

0000009C This is the ERRNO parameter. 9C means *EMVSINITIAL Process initialization error*, which indicates a problem during initialization of TCP/IP for MVS as an OpenEdition process.

0B0C00FB This is the reason code parameter. The last four hexadecimal digits is the actual reason code. It means *User is not authorized to OpenMVS*.

The TCPUSER value is the started task user ID of the TCP/IP system address space. This message tells us that this RACF user ID did not have the proper

definitions to allow the TCP/IP system address space to use OE services. The specific problem in this situation was that the TCPUSER user ID did not have the proper OpenEdition definitions in RACF; we had not defined an OMVS RACF segment (see 1.4, “OpenEdition User Identification” on page 14).

1.3 OpenEdition Processes and Threads

In order to set up, configure, and possibly debug OpenEdition application programs, such as the TCP/IP for MVS OpenEdition Applications Feature or the Internet Connection Server for MVS, you need to have a basic understanding of the process model that is used within the OpenEdition environment.

In MVS we have the following basic categories of work:

- Started tasks (MVS operator start command)
- Batch jobs (submit via JES)
- TSO/E user (logon)
- APPC/MVS transactions (CPI-C allocate)

All work that is created through one of the above requests finally ends up in an address space that holds every piece of information that is required to describe the work at any moment in time (for example, storage control blocks, program(s) to execute, opened data sets, etc.). Though representing the current work, the address space is not the unit of work MVS dispatches. MVS dispatches a TCB (task control block), an IRB (interrupt request block), or an SRB (service request block). They are all dispatchable units that represent work that runs in an address space or runs on behalf of work in an address space.

Basically this does not change in the OpenEdition environment, but we work with a slightly different terminology that is based on the concepts of a *process* and a *thread*.

1.3.1 OpenEdition Process

A process maps to an MVS address space and an MVS task environment exists for the process, in terms of a task control block (TCB) and related control blocks. In addition to the TCB, the OE Kernel address space maintains a number of control blocks that represent an OpenEdition process. These control blocks exist within the OE Kernel address space and are created whenever an existing standard MVS program begins using OpenEdition services (the MVS address space is being *dubbed* as an OpenEdition process) or a new OpenEdition process is being created by the OpenEdition process creation functions. By default, a new process will run as the job step task in an MVS address space. There are situations where more processes may exist within the same MVS address space, and in such a case a process may be running as a subtask of the job step task.

An OpenEdition program may use a number of OpenEdition services to create new processes or to enable multithreading within the process itself. It should be pointed out that there are no means to prohibit creation of new processes by an application programmer. See Figure 4 on page 10 for an overview of the process-related services of OpenEdition.

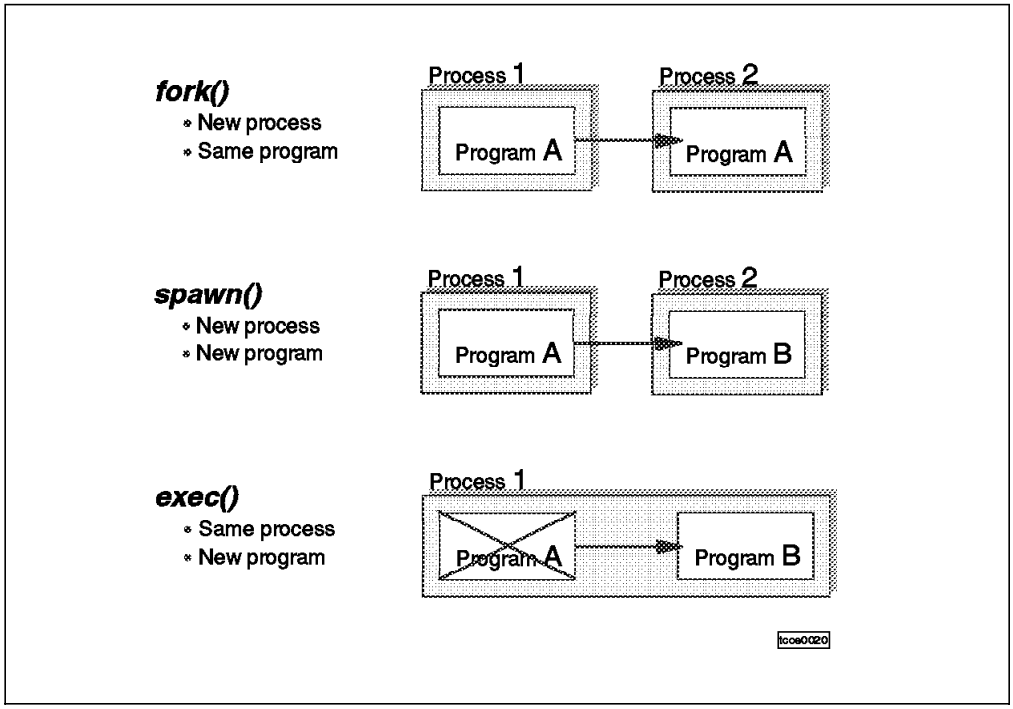


Figure 4. Process-Related Services

To control processes, the following basic services are available:

fork The fork() service basically replicates the current process into a child process. After the fork() call, the two processes are identical and both parent process and child process continue as independent processes.

spawn The spawn() service also starts a new process, but the child process is started with another program as indicated by the parent process on the spawn() call. After the spawn() call, the two processes continue as independent processes.

exec The exec() service does not start a new process, but replaces the program in the current process with another program as indicated on the exec() call.

1.3.1.1 Forking a New Process

To start a new process, a process may use the fork() service. Forking is a very well-known concept in UNIX environments, but it is not a function that directly maps into any traditional MVS system services. See Figure 5 on page 11 for an introduction to the fork() service.

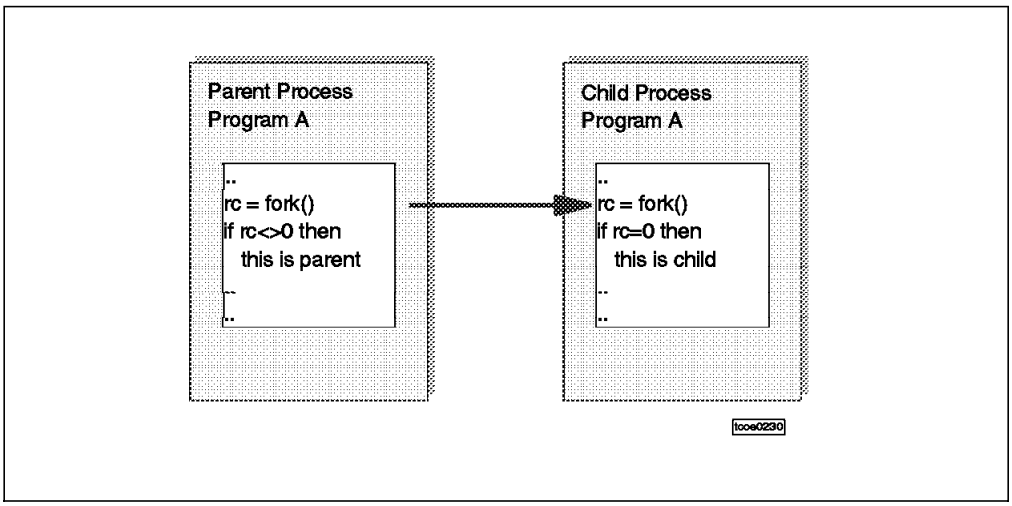


Figure 5. The Fork Service

If you have a process running that is executing, for example, program A and this program calls the fork() function, the OE Kernel address space initiates the following actions:

1. Invokes APPC/MVS services to schedule a new address space for a new process.
2. Moves user recovery routines and contents supervisor structures from the parent process address space to the child process address space. Storage is made logically available in such a way that read access to pages from the child will be backed by the same frames the parent process has. With the first write access by the child to a page, a new frame is allocated for the child process.
3. Returns control to the instruction following the fork() call in *both* processes.

Just after a fork() call, the two processes are almost identical; the program is the same, they have access to the same storage, the user security environment is the same, and any Hierarchical File System file descriptors or socket descriptors that were opened by the parent process are also open and available to the child process. Positions established by the parent process in sequentially processed files before the fork() call are maintained and preserved in the child process. In fact, the only difference between the two processes is the return code from the fork() call; the parent process receives the process ID (PID) of the child process, while the child process receives a return code of zero. It is important to understand that control is given to the child process at the instruction following the fork() call and not at the program′s main entry point, as you, based on your traditional MVS experience, might have expected.

We need to point out one important aspect of this inheritance concept that applies to DD-name allocations of any kind. If the parent process had made a DD-name allocation (either via JCL, TSO ALLOC or dynamic allocation services) before the fork() call, this allocation is *not* inherited by the child process. This exception is important to note when we begin to configure our TCP/IP for MVS system for OpenEdition use. One of the techniques we have available for pointing to a TCPIP.DATA configuration data set is to use the SYSTCPD DD-name allocation. Let us emphasize that this is not a good technique in the OpenEdition environment, because forking takes place all the time. If you allocated your TCPIP.DATA configuration data set to a process via a DD-name allocation, that

allocation is not available to child processes that are forked off from your parent process. The only exception to this rule is a STEPLIB allocation. If your process had started with a STEPLIB DD allocation, this allocation will be passed to the child process.

In most implementations, the parent process will go on doing what it has to do, and the child process will most likely do cleanup and pass control to a child-specific program that will do whatever the child process has to do.

1.3.1.2 Spawning a New Process

Another mechanism to start a new process is to use the spawn() service. This service works very much like a fork() service, except for the fact that the new process is not a copy of the parent process. On the spawn() call the parent process specifies the name of the program to start in the child process, and the new process is started with this new program being given control at its main entry point.

As for the fork() service, the child process inherits file and socket descriptors, but these have to be re-opened by the child process in order for the child process to use them. Any position in files that were processed sequentially by the parent process are lost and the position is reset to the start of these files when the child process re-opens the files. A spawned process does not inherit any DD-name allocations. The spawn() service has the same restrictions on using DD-name allocations as the fork() service.

The parent process is able to control if a spawn() call will result in a process being started in another address space or as a task within the same address space as the parent process itself. This is controlled by setting proper environment variables. If the environment variable _BPX_SHAREAS has been set to YES before the spawn() call is initiated, the child process will start within the same address space as the parent process and APPC/MVS will not be involved in the initialization of the new process.

Some applications allow you to set a configuration variable that is used by the application to control whether new processes are started within the same address space or within a new address space. Where such configuration options are available, it is generally a good idea to turn them on. Starting a new process within the same address space as the parent process requires much less processing and will in general perform better than starting the new process in another address space. There are restrictions on when it is possible to do so. Please see the OpenEdition programming reference manuals for details on those restrictions.

1.3.1.3 Replacing the Program in a Process

If a program in a process wants to replace itself with another program, it can use the exec() service. This service will preserve the current process environment, but completely replace the program that is running within that process. A successful exec() call will never return control to the calling program, but control will be passed to the main entry point of the new program that is specified on the exec() call.

The exec() service is typically used after a fork() by the child process to replace the parent process program with a child-specific program to perform the child-related functions of the application.

1.3.2 OpenEdition Threads

If a program within an OpenEdition process needs to work with more dispatchable units of work, but does not need the advanced functions of the fork() or spawn() service, it can use the POSIX threading services that are part of the OpenEdition component.

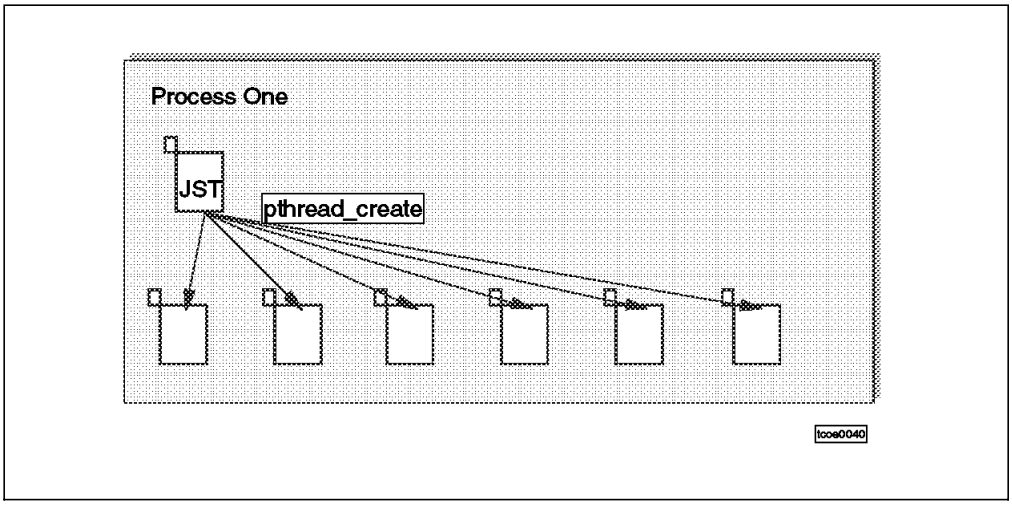


Figure 6. Overview of POSIX Threads in OpenEdition

The application program can create and manage threads via a range of special threading system service calls. To create a new thread, an application program will use the pthread_create() service.

There are, in general, three types of threads:

1. Light-weight threading

   Light-weight threads are not assigned to individual subtasks, but are managed within one task. This is also sometimes referred to as pseudo-subtasking. OpenEdition does not implement light-weight threads.

2. Medium-weight threading

   With medium-weight threading services, the program allocates a pool of MVS subtasks to be used for threads. When a new thread is required, one of the free subtasks within the pool is used. If no subtask is available, the thread creation is postponed until a subtask becomes available. With medium-weight threading it is important to monitor how often a thread creation is postponed because there are no available subtasks. If this happens often, it might be a good idea to increase the size of the subtask pool that is created by this application. An example of an OpenEdition application that uses medium-weight threading is the Internet Connection Server for MVS. One of the configuration parameters of this server is the number of subtasks in the pool (the MinActiveThreads directive).

3. Heavy-weight threading

   If the application uses heavy-weight threads, a new MVS subtask is created every time the application requests a new thread to be created. The subtask is terminated when the thread terminates.

OpenEdition resources, such as Hierarchical File System files, sockets, pipes etc. are available to all threads within a process.

There is a high similarity between POSIX threads and MVS subtasking except for the following: MVS subtasks are normally used to run a piece of code that may run on its own (a separate load module), while the pieces of code that are running on POSIX threads are part of one and the same load module.

1.4 OpenEdition User Identification

All users of an MVS system, including users of OpenEdition functions, must have a valid MVS user ID and password. To use standard MVS functions, the user must have the standard MVS identity based on the MVS user ID and group name.

If a unit of work in MVS uses OpenEdition functions, this unit of work must, in addition to a valid MVS identity, have an OpenEdition identity. An OpenEdition identity is based on a UNIX user ID (UID) and a UNIX group ID (GID). Both UID and GID are numeric values ranging from 0 to 2147483647. In an OpenEdition system, the UID is defined through the OMVS segment in the user′s RACF user profile, and the GID is defined in an OMVS segment in the group′s RACF group profile. What we in an MVS environment call the user ID is in a UNIX environment normally termed the user name or the login name. It is the name the user uses to present him or herself to the operating system. In both an OpenEdition system and in other UNIX systems, this user name is correlated to a numeric user identification, the UID, which is used to represent this user wherever such information has to be stored in the OpenEdition environment. One example of this is in the Hierarchical File System, where the UID of the owning user is stored together with each individual file.

Access to resources in the traditional MVS environment is based on the MVS user ID, group ID and individual resource profiles that are stored in the RACF database.

Access to OpenEdition resources is granted *only* if the MVS user ID has a valid OMVS segment with an OMVS UID. Access to resources in the Hierarchical File System is based on the UID, the GID and file access permission bits that are stored with each file.

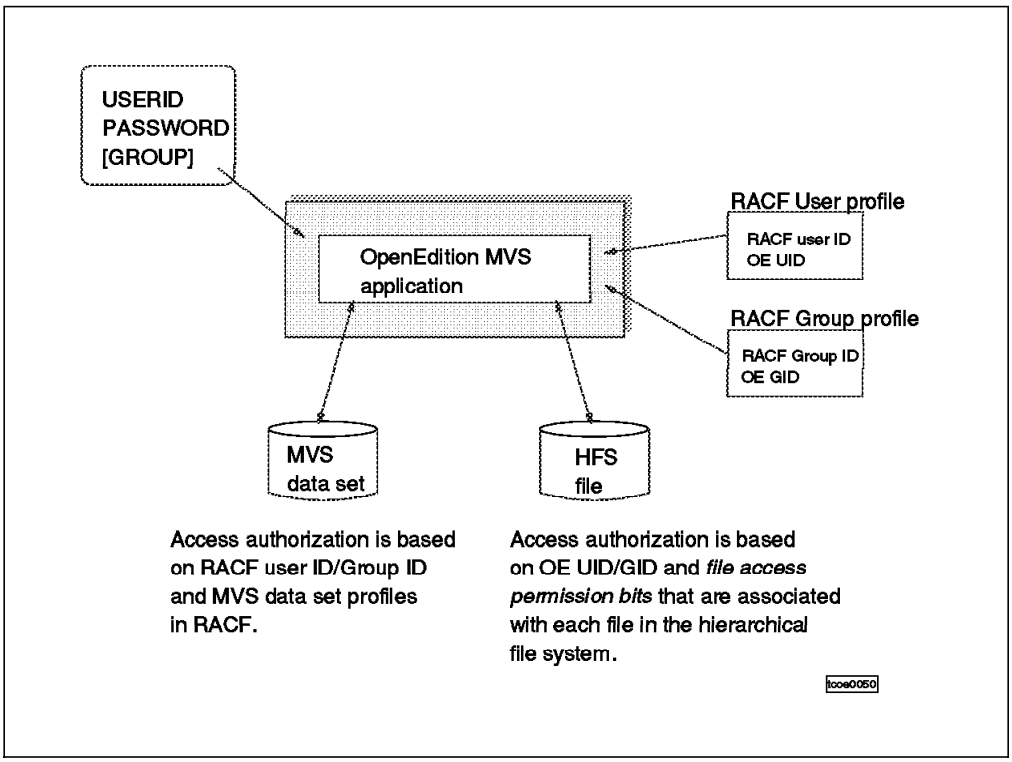


Figure 7. Resource Access Authorization

When a process issues its first call to an OpenEdition service, it is said to be *dubbed*; it is given an OpenEdition identity that is based on the OMVS segments of the associated RACF user and group profiles.

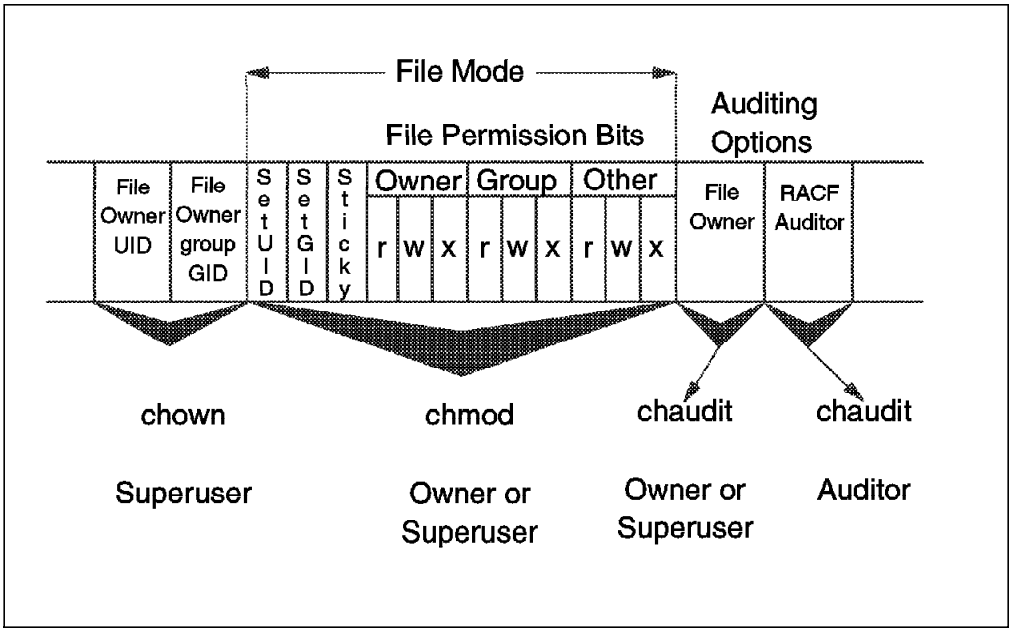


Figure 8. Hierarchical File System File Security Packet

The basic access algorithm can be derived from Figure 8. Every file in the Hierarchical File System has a file security packet (FSP) assigned to it. This FSP contains the identification of the owning UID and GID along with information

about what level of access (read, write, execute or in short form: rwx) is granted to the three user categories that are considered to exist in this environment:

- The owner. Please note that the owner of a file may restrict him or herself to read and execute (r-x) access.
- The other members of the owning group.
- Everyone else. Also generally known as *world access*.

When accessing a file in the Hierarchical File System, the file system looks up the current process UID and GID and compares these to the owning UID and GID. If this fails, the check proceeds under the assumption of world access. These checks are all done in cooperation with RACF. Consequently it is not surprising that in case all checks fail, you will see an appropriate message in the MVS system log as shown in Figure 9.

```
ICH408I USER(HDM     ) GROUP(OMVSGRP ) NAME(HANS MERTIENS       ) 894
  /u/alfredc/oec.c CL(FSOBJ   ) FID(01E2D4E2E5F0F2000110000000000003)
  INSUFFICIENT AUTHORITY TO OPEN
  ACCESS INTENT(-W-)  ACCESS ALLOWED(GROUP --X)
```

Figure 9. Resource Access Authorization Failure

For a detailed description of how security is implemented in the OpenEdition environment please see *RACF Support for Open Systems*, GG24-2005.

1.5 UID/GID Assignment to a Process

As we mentioned earlier, the first attempt to use OMVS services *dubs* the MVS address space into an OpenEdition process. This adds new information to the current address space, of which the UID/GID assignment probably is the most important:

Real UID At process creation, the real UID identifies the user who has created the process.

Effective UID Each process also has an *effective UID*. The effective UID is used to determine *owner* access privileges of a process.

Normally this value is the same as the real UID. It is possible, however, for a program that resides in the Hierarchical File System to have a special flag set that, when this program is executed, changes the effective UID of the process to the UID of the owner of the program. A program with this special flag set is said to be a *set-user-ID* program. This feature provides additional permissions to users while the set-user-ID program is being executed.

Saved UID Used to save the effective UID of a process. When an OpenEdition MVS user tries to change the UID, the saved_UID field is checked to see if the UID is the original one.

Real GID At process creation, the real GID identifies the group of the user for which the process was created.

Effective GID Each process also has an effective group. The effective GID is used to determine *group access* privileges of a process.

Normally this value is the same as the real GID. A program can,

however, have a special flag set that, when this program is executed, changes the effective GID of the process to the GID of the owner of this program. A program with this special flag set is said to be a *set-group-ID* program. Like the set-user-ID feature, this provides additional permission to users while the set-group-ID program is being executed.

Saved GID Used to save the effective GID of a process. When an OpenEdition MVS user tries to change the GID, the saved GID is checked to see if the GID is the original one.

The real UID/GID tells us who we really are; the effective UID and GID are used for file access permission checks; the saved values of UID and GID are stored by the exec() function.

See Table 2 for the relations between the values described above and how they are manipulated by various function calls.

Note: Although we only reference the setuid() function, the same applies to the GID as handled by the setgid() function.

Table 2. Summary Showing How the UID May Be Changed

ID	exec()		setuid()	
	set-uid-bit off	**set-uid-bit on**	**superuser**	**normal user**
real user ID	unchanged	unchanged	set to UID	unchanged
effective user ID	unchanged	set from owner UID of program file	set to UID	set to UID
saved set-user ID	copied from effective UID	copied from effective UID	set to UID	unchanged

Notes:

1. exec() indicates the program execution handler.
2. setuid() indicates the call to the C language function.

1.5.1 Superuser

One UID value has a special meaning in the world of UNIX and consequently it has this meaning in the OpenEdition environment. A user that has a UID of zero may access *every* resource in the UNIX environment and therefore in the OpenEdition environment as well. This user is said to be a *superuser*. Even if the superuser may not access a file due to missing group or world access, he/she may redefine the access rights using a chmod command. Doing this will immediately give the superuser whatever access he/she wants to have.

In most UNIX systems, a UID of zero gives unlimited right to access and manipulate files in the Hierarchical File System and to change the identity of the process that is running with a UID of zero. In most UNIX systems this means that the system administrator normally has a UID of zero, and any server programs that need to change the identity of forked processes run with a UID of zero.

You can choose to run your OpenEdition environment based on the standard meaning of a zero UID, but you are also given the option of defining your security setup in the OpenEdition environment, so you can distinguish between and control different levels of superuser privileges.

If you just define the user IDs under which your OpenEdition servers are started with a UID of zero and do not create any BPX RACF facility class definitions, your system runs as any UNIX system would run. A UID of zero means that the process has full access rights over all files in the Hierarchical File System and that the process can change OpenEdition user identity as required by, for example, server programs. You have to be very careful how you assign such a user ID to your processes. Any process running with a UID of zero can delete your root file system and change identity to any of your defined users.

You may use another approach in an OpenEdition system, where you can define three RACF facility classes to help you distinguish between different levels of superuser privileges. The three facility classes are:

BPX.SUPERUSER This facility class can be used to allow users with a non-zero UID to change the effective UID of their process into a UID of zero. If you have users who occasionally, as part of their daily system work, need to have superuser privileges, you can assign them a non-zero UID and permit them use of BPX.SUPERUSER. The alternative would have been to define all such users with a permanent UID of zero, which meant that they performed all their work as superuser, even work that did not require them to be superusers. By using this facility class you can limit the amount of time any user runs in superuser mode. Remember that in superuser mode, the user can, perhaps by mistake, delete the root file system of your Hierarchical File System. If a user who is permitted access to BPX.SUPERUSER logs on to the OpenEdition interactive shell, the user can type in an su command to switch into superuser mode, and an exit command to return to his normal non-zero UID environment.

BPX.DAEMON User IDs that are assigned to server processes that need to change user identity must be permitted to this facility class. As soon as the BPX.DAEMON facility class has been defined in RACF, a UID of zero is not sufficient for changing the user identity of a process. The process must run with a UID of zero *and* must be permitted to BPX.DAEMON to do so. Typically server processes (for example, the telnet server (TelnetD), the remote execution server (REXECD), and the file transfer server (FTPD)) need to be able to change the OpenEdition identity to the identity of the client user after having obtained a valid user ID and password. There are a number of configuration tasks that you must perform if you choose to use this level of security in your environment. Please see Chapter 2, “Security” on page 49 for details of these tasks.

BPX.SERVER This facility class is used in addition to a UID of zero and a permission to BPX.DAEMON. It allows for a two-level authority check to use a surrogate user ID. A server address space user ID may have either READ or UPDATE permission to BPX.SERVER:

UPDATE Application servers that are permitted with UPDATE access to this profile may act as a surrogate of the client. There is an additional security check performed to determine if this server has SURROGATE authority for the client user ID. This environment is also called an *authenticated client*. In this environment *only* the client′s RACF identity and authorizations are used in resource access decisions processed by RACF. The Internet Connection Server for MVS requires this level of authorization.

READ If the application servers are permitted with READ access to this profile, two identities are used in local control decisions:

- The RACF identity of the client
- The RACF identity of the server

This environment is called an *unauthenticated client*.

As we already mentioned, the RACF OMVS segment is the place where the major UNIX type characteristics of a user ID are defined. These characteristics are:

UID The user′s UID, which may be in the range of 0 to 2147483647.

HOME The user′s default home directory in the Hierarchical File System. This value can be found in the environment variable HOME.

PROGRAM The initial program to start when the user logs on to the interactive shell. This is normally specified as /bin/sh, which is the default interactive shell program.

When the user logs on to the interactive shell, the HOME and PROGRAM environment variables are initialized based on the corresponding RACF attributes in the user′s RACF OMVS segment. The environment variable LOGNAME is set to the user′s RACF user ID (the login name).

See Chapter 2, “Security” on page 49 for the configuration activities you have to perform in order to enable OpenEdition server security.

1.6 Sockets in the OpenEdition Environment

Any program that uses a socket application programming interface must be compiled and linked with a socket library. A socket library consists of one or more of the following components:

- Header files, include files, or copy structures that are used during compilation of the socket application program. These files define commonly used data structures and interfaces to socket-related functions.
- Link library with object modules to be statically linked with the socket application program during linkage editing processing.
- Run-time library with run-time programs to support the socket calls that are being used by the application program.

1.6.1 Sockets in MVS

In an MVS environment you have a number of socket libraries that are available for your use. Most socket libraries are language-specific. The most common socket programming language is C, but other programming languages are supported, too.

TCP/IP for MVS supplies the following main socket libraries:

- TCP/IP for MVS C-sockets.
- TCP/IP for MVS sockets extended - callable sockets.

  This library can be used with application programs that are written in COBOL, PL/I or assembler. All socket functions are implemented via a standard call interface.
- TCP/IP for MVS sockets extended - assembler macro sockets.

  This library can be used from assembler programs. The socket interface is implemented via a standard assembler macro interface.
- TCP/IP for MVS REXX sockets.

  This library is used from REXX programs and the socket functions are implemented through a standard REXX environment that is used in an address SOCKET statement.
- TCP/IP for MVS CICS sockets.

  This socket library is used to allow CICS application programs in CICS tasks to use socket functions. The CICS socket APIs include a limited C-socket API and the sockets extended callable API to be used from COBOL, PL/I and assembler CICS programs.
- TCP/IP for MVS IMS sockets.

  This socket library is used to allow IMS application programs to use socket functions. The supported API is the sockets extended callable API to be used from COBOL, PL/I and assembler IMS application programs.

Common to all these TCP/IP for MVS socket libraries is that they use the TCP/IP for MVS techniques to pass data between the socket application program address space and the TCP/IP system address space.

If you write your socket program in C, you have more choices available in selecting the C-socket library you want to use. In an MVS environment, you currently have no less than three different C-socket libraries you can choose from:

1. The TCP/IP for MVS C-socket library
2. The AnyNet MVS C-socket library
3. The OE socket library

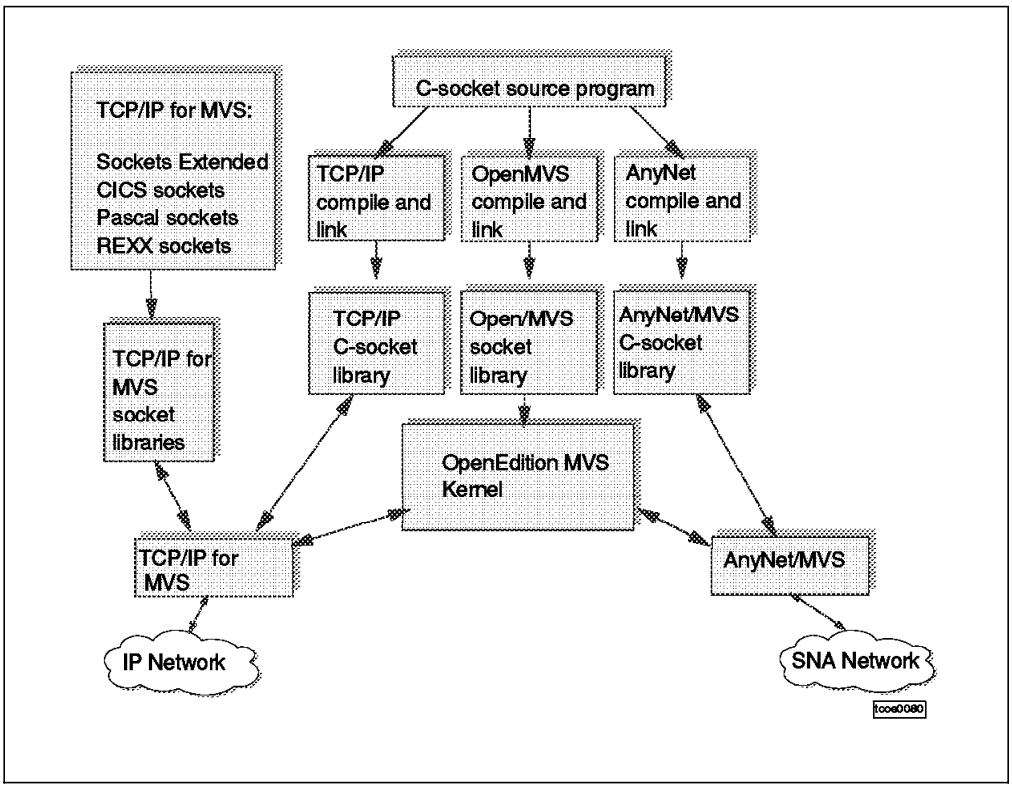


Figure 10. Socket Libraries in MVS

1.6.2 TCP/IP for MVS and AnyNet MVS C-Sockets

The TCP/IP for MVS C-socket library and the AnyNet MVS C-socket library more or less support the same socket calls. In most situations, it is possible to compile and link the same source code with both socket libraries. If you write your C-socket program to be used with one of these two libraries, you can compile and link it with both these libraries. It is important to emphasize that a C-socket program is compiled and linked with *one* of these libraries. If the program is compiled with the socket library from TCP/IP for MVS, the program works with TCP/IP for MVS, but not with AnyNet MVS. If the program is compiled and linked with the socket library from AnyNet MVS, the program works with AnyNet MVS, but not with TCP/IP for MVS. If you want a socket program to work with both TCP/IP for MVS and AnyNet MVS, you need to compile and link the source code into two separate load modules: one for use with TCP/IP for MVS and another for use with AnyNet MVS.

If your socket program is a server program that is supposed to serve clients that connect through both TCP/IP for MVS and AnyNet MVS, you have to start two instances of the server program: one address space that runs the TCP/IP for MVS version and another server address space that runs the AnyNet MVS version of the program. The same executable copy of the program cannot work concurrently with TCP/IP for MVS and AnyNet MVS when you compile and link it with the TCP/IP for MVS and AnyNet MVS C-socket libraries.

1.6.3 OpenEdition Sockets

When you write socket application programs to be used with the C-socket libraries from TCP/IP for MVS and AnyNet MVS, you have to use a separate set of C functions to access files and another set of C functions to access sockets. To read data from a socket, you can use the read() call, and to read data from a file, you can use the fread() call.

A C program works with handles that represent resources, such as files or sockets. Such a handle is called a *descriptor*, and is a 16-bit integer that holds a descriptor number, which represents the resource in question. In a C program that uses the TCP/IP for MVS or AnyNet MVS socket libraries, one part of the run-time environment manages the descriptors that represent, for example, files (file descriptors), and another part, the socket library, manages the descriptors that represent sockets (socket descriptors). In such a program, there is no common management of descriptor numbers, which is why the programmer has to use different functions for accessing a file or a socket. The C run-time environment is not able to determine if a descriptor is a file descriptor or a socket descriptor, so the programmer must indicate that via the functions that are used to access the resources.

In a POSIX-compliant program, the programmer does not have to make such a distinction. In such a C program, the programmer uses the same functions to access both files, pipes, sockets and other resources that are used by the C program. The C run-time environment must therefore be able to determine dynamically what kind of a descriptor is passed on the individual function calls, it must have information available for all the descriptors that are in use and manage assignment of all new descriptors across all the supported resource types.

In OpenEdition this is accomplished through the use of the following components, as shown in Figure 11 on page 23:

1. An OpenEdition C-socket library
2. A component that is called the logical file system (LFS)
3. A set of components that are called physical file systems (PFS)

All C calls that use descriptors are passed to the logical file system. The logical file system manages assignment of new descriptors and maintains information about the type of resource that is represented by the individual descriptors.

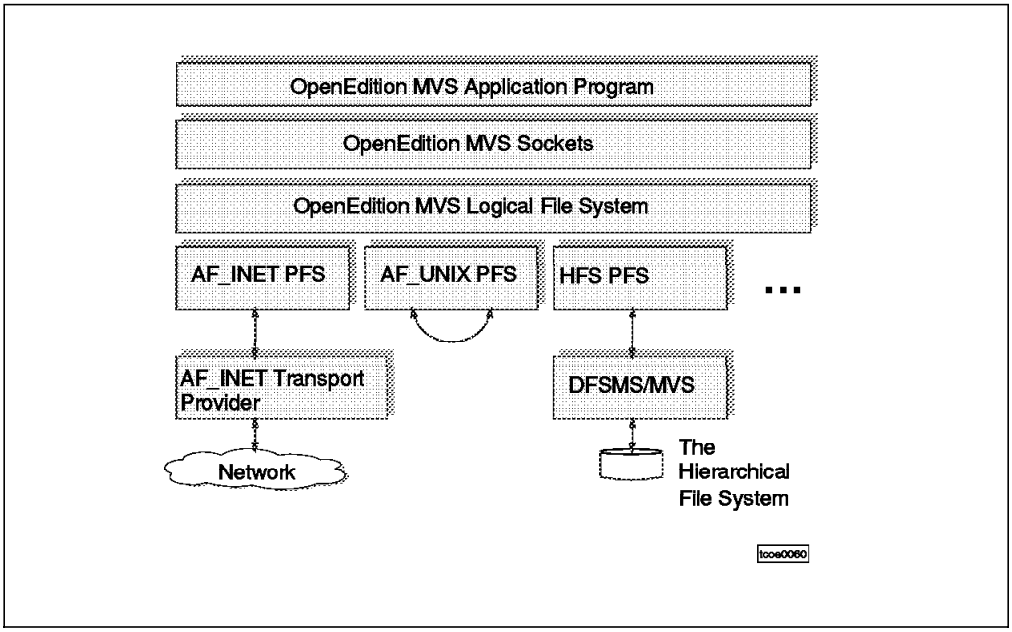


Figure 11. Sockets in OpenEdition

Based on the type of resource, individual function calls are passed to the associated physical file system for execution. There are a number of physical file systems in OpenEdition. The most common physical file systems are:

- The Hierarchical File System PFS.

  This PFS takes care of requests that are related to resources in the Hierarchical File System, such as traditional files or special character files.

- The AF_UNIX PFS.

  If the descriptor represents an AF_UNIX socket, the request is handled by this PFS. AF_UNIX sockets are so-called local sockets that can be used by two OpenEdition application programs on the same OpenEdition system to communicate with each other.

- The AF_INET PFS.

  If the descriptor represents an AF_INET socket, the request is handled by this PFS. An AF_INET socket is what is used on a TCP/IP-based network and is generally known as a network socket. If an OpenEdition application program wants to communicate with a socket program on a TCP/IP host on an attached IP network, the program will open an AF_INET socket for that purpose.

From a TCP/IP point of view, it is the AF_INET physical file system that has our main interest.

MVS/ESA SP 5.1 introduced the integrated sockets AF_INET PFS. In MVS/ESA SP 5.2.2 another AF_INET physical file system was introduced, which is the converged sockets PFS, also called the Common I-Net PFS, or for short C-INET. In an MVS/ESA SP 5.2.2 or OS/390 system, you can choose which of the two AF_INET physical file systems you want to use.

1.6.4 Integrated Sockets PFS

The integrated sockets PFS supports one AF_INET transport provider. This transport provider can either be a TCP/IP for MVS stack or it can be an AnyNet MVS stack, but only one stack at a time is supported. All AF_INET socket calls are handled by this one AF_INET transport provider, as shown in Figure 12.

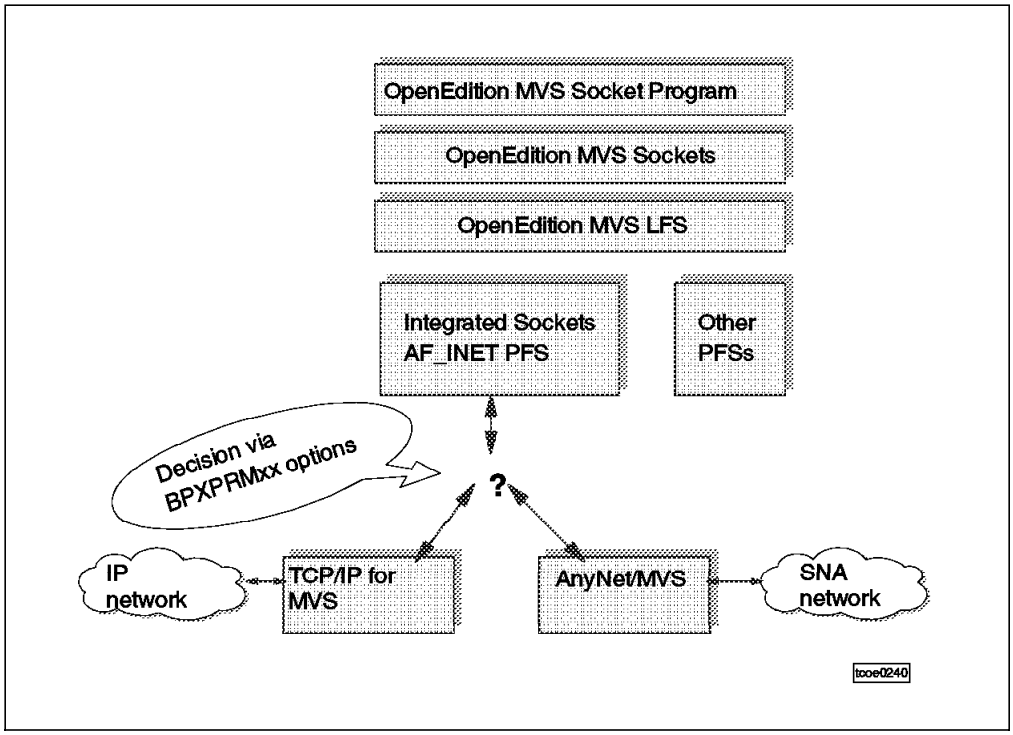


Figure 12. Integrated Sockets

You specify in the OpenEdition BPXPRMxx SYS1.PARMLIB member if you want TCP/IP for MVS or AnyNet MVS as the AF_INET transport provider. If you start more TCP/IP for MVS stacks on your MVS system, you can not use the BPXPRMxx parmlib member to specify which of the TCP/IP for MVS stacks you want as AF_INET transport provider. The first TCP/IP for MVS stack that connects to the OE Kernel address space becomes the AF_INET transport provider. You can control, via parameters in your TCP/IP for MVS PROFILE configuration data set, if a TCP/IP for MVS stack should try to connect to the OE Kernel address space as a transport provider. By default, any TCP/IP for MVS stack will try to connect to OpenEdition. You can prevent a stack from doing so by entering the NOOE configuration parameter in that stack′s PROFILE configuration data set:

```
; ************************************************************
; * Do NOT attempt to connect to OE from this TCP/IP stack     *
; ************************************************************
;
NOOE
;
```

You have a similar configuration option in the AnyNet MVS environment parameter data set. The default for an AnyNet MVS stack is to try to connect to OpenEdition, but you can prevent an AnyNet MVS stack from doing so by coding the following in the AnyNet MVS environment parameter data set:

```
# ************************************************************
# * Do NOT attempt to connect to OE from this AnyNet stack     *
# ************************************************************
#
OPEN_EDITION=NO
#
```

1.6.5 Converged Sockets PFS

In MVS/ESA SP 5.2.2 and in OS/390 the converged sockets AF_INET physical file system was introduced. This PFS allows more AF_INET transport providers to be connected to the OE Kernel address space at the same time, supporting concurrent access from more AF_INET stacks, for example, a TCP/IP for MVS stack and an AnyNet MVS stack, as shown in Figure 13.

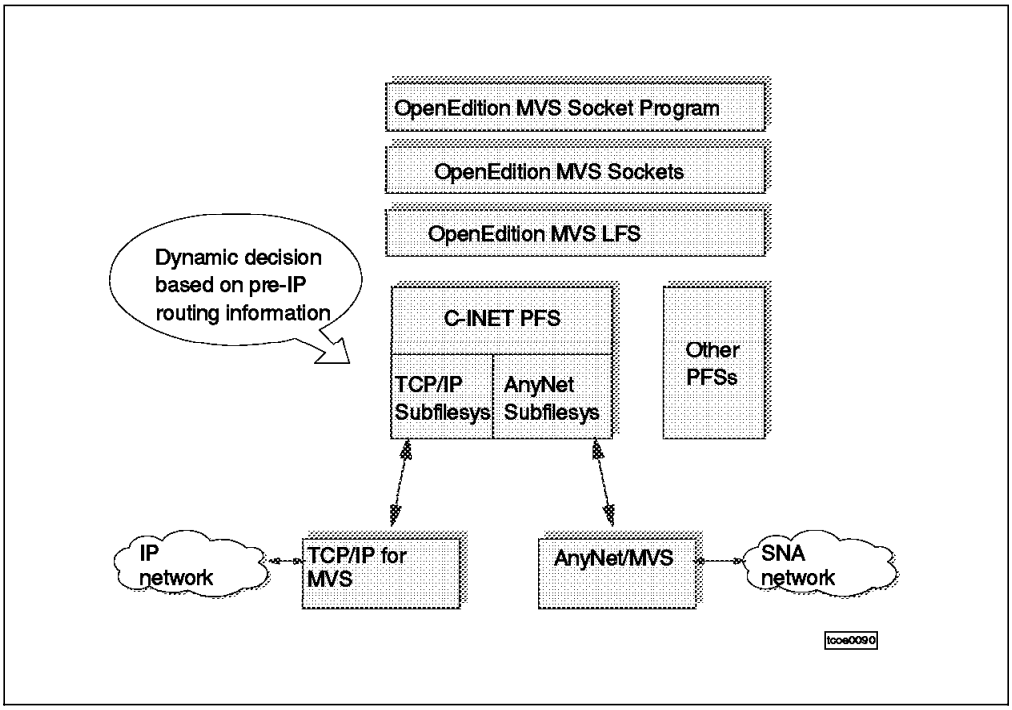


Figure 13. Converged Sockets

One instance of an OpenEdition AF_INET socket program that runs in the OpenEdition environment can concurrently service client requests that arrive over a TCP/IP for MVS stack and requests that arrive over an AnyNet MVS stack.

The converged sockets PFS includes a so-called *pre-router*, which determines how individual socket calls are passed to the connected transport providers. Some calls are routed directly to a single transport provider while other calls are propagated across all the connected transport providers. In addition to the pre-router, the converged sockets PFS consists of one subfilesystem per connected AF_INET transport provider.

An OpenEdition AF_INET socket program does not have any knowledge of the existence of one or more transport providers. From an OpenEdition socket program point of view, there is one AF_INET stack with a number of network interfaces. In reality, the individual network interfaces may be associated with different stacks, but the OpenEdition socket program does not know that. The

converged sockets PFS gives the appearance of a single converged TCP/IP stack.

Each transport provider has a separate set of interfaces, separate networking layer with IP and ICMP, and separate transport protocol layer with TCP and UDP. But the application layer is considered to be shared among all connected stacks, as shown in Figure 14.

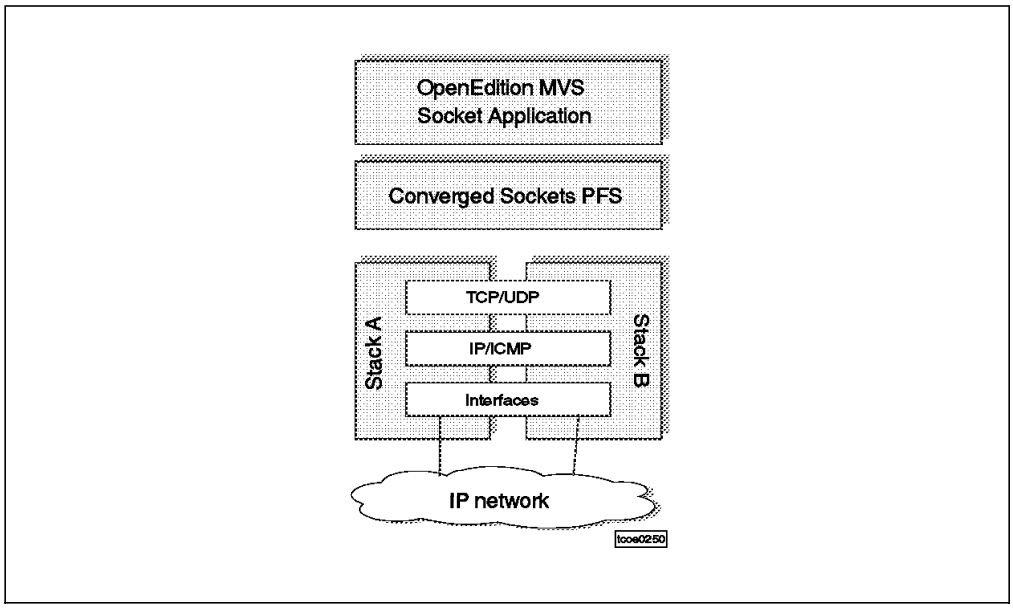


Figure 14. Converged Stacks As Perceived by OpenEdition Applications

When a remote socket client connects to an OpenEdition server program, that connection request arrives over one of the AF_INET transport providers, and all socket calls from the OpenEdition server program for that connection are routed directly to that one AF_INET transport provider. But there are socket calls, that can not be routed to a single transport provider. One example of this is a stream socket server program (TCP protocols) that starts up in the OpenEdition environment. Such a server program issues a number of initial socket calls that establish the program as a server program that will accept requests from clients on the connected internets. The sequence of calls is:

1. socket() - Open a socket. This call is propagated to all connected AF_INET transport providers in order to open a socket in each transport providers transport protocol layer.
2. bind() - Bind the socket to a local server port number. This call is propagated to all connected AF_INET transport providers in order to establish the server program′s identity on all stacks.
3. listen() - Prepare to receive client connection requests. This call is propagated to all stacks in order to signal the server program′s intent to receive client requests over all connected transport providers.
4. accept() - Wait for the next client to connect. As the server accepts client requests over all connected transport providers, this call is also propagated to all transport providers.

When a client actually connects through one of the transport providers, the succeeding socket calls for that one connection are routed only to that one stack over which the connection was established.

Consider another type of socket program: a client program that starts in the OpenEdition environment. If this program is a stream socket client program, it issues a connect() call where the client program specifies the internet address of the server host and the port number on which the server program is running on that destination host. When this request comes down to the converged sockets PFS, the pre-router must determine which of the connected transport providers have the best route to the requested destination IP address. This decision is made based on a copy of each transport providers IP layer routing table. When a stack connects to the converged sockets PFS, the pre-router queries the newly connected stack for a copy of its IP layer routing table. Each time the transport provider updates its IP layer routing table, the pre-router receives a signal from the transport provider and initiates a process to obtain a new copy in order to keep an up-to-date accurate copy of all the destinations that are supported by the connected transport providers.

A stack may update its IP layer routing table in more ways. It may do so as the result of a manual update of the routing tables, for example, via a TCP/IP for MVS OBEYFILE command that replaces the static route definitions for a TCP/IP for MVS stack, or for an AnyNet MVS stack via an execution of the ISTSKRTE utility program to manually update the AnyNet MVS IP layer routing table. IP layer routing tables in TCP/IP for MVS may also be updated via ICMP redirects or via a dynamic update from the RouteD server program based on new route information received from other RouteD or GateD servers on the connected IP networks.

It is important to emphasize that the Common I-Net pre-router does not make any IP level routing decisions. The pre-router only uses its copy of the IP layer routing tables to *select* an appropriate stack for certain socket calls. When the transport protocol layer in the selected stack has constructed an IP datagram and passed this IP datagram to that stack′s IP layer, normal IP layer routing decisions are being made by that IP layer based on its IP layer routing table. Please also note that this means there is no IP layer routing taking place via the Common I-Net PFS between stacks. If we have two stacks connected to the Common I-Net PFS, these two stacks cannot route an IP datagram between them via the Common I-Net PFS. If the two stacks need to route IP datagrams between them, they will either have to use an IUCV link between them (if both stacks are TCP/IP for MVS stacks) or they will have to be connected to the same physical IP network.

Please see Chapter 4, “Setting Up the AF_INET Transport Providers” on page 83 for detailed instructions on how you set up various AF_INET transport provider configurations that include both TCP/IP for MVS and AnyNet MVS.

1.7 OpenEdition Resolver Configuration Data

One of the components you will find in a socket library is the *resolver*. The resolver in the OpenEdition socket library is delivered as part of the C run-time library and is used by all OpenEdition socket programs. The function of the resolver is to service a number of application calls to obtain, for example, the local port number for a given server program or to resolve an IP host name into one or more IP addresses. Some of the calls to the resolver are serviced via looking up information in a number of local configuration data sets or files, while other calls are serviced by sending requests to remote server programs, generally known as domain name servers.

All socket libraries have a resolver component. This also means that if you have a TCP/IP for MVS stack and an AnyNet MVS stack connected to your OpenEdition environment, you have three different resolver components, each requiring a set of resolver configuration data sets or files, as shown in Figure 15 on page 28.

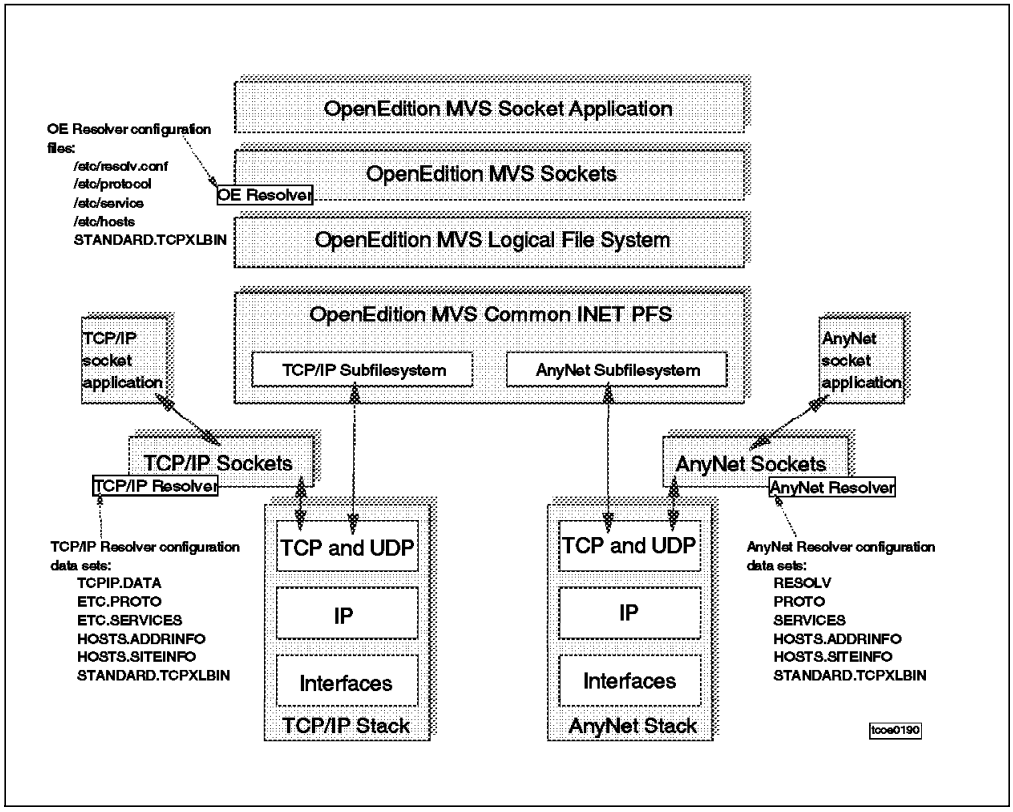


Figure 15. Resolver Components and Related Configuration Information

The main configuration data set for any resolver function is the resolver configuration data set or file. In a UNIX system, this file is normally located in the /etc/resolv.conf file. In a TCP/IP for MVS system, the resolver configuration data set is called TCPIP.DATA and can be located various places of which the most commonly used is SYS1.TCPPARMS(TCPDATA). In an AnyNet MVS environment, the resolver configuration data set is pointed to via the AnyNet MVS environment data set. The keyword is RESOLV, and it points to a fully qualified MVS data set. In general, AnyNet MVS is able to work with the same resolver configuration data set formats as TCP/IP for MVS is using.

The OpenEdition resolver needs access to the following information:

- Resolver configuration
- Protocols supported
- Services supported
- Locally known host names
- ASCII-EBCDIC translation table for the resolver

1.7.1 Resolver Configuration Data

The contents of this configuration data set or file are compatible with the TCP/IP for MVS TCPIP.DATA configuration data set. The only parameter in an existing TCPIP.DATA configuration data set that has no meaning in an OpenEdition environment is the TCPIPJOBNAME or TCPIPUSERID keyword. An OpenEdition socket program does not contact the TCP/IP system address space directly, but leaves that communication to take place in the physical file system component. The AF_INET PFS uses other techniques to decide which TCP/IP system address space to use for a given socket call.

The search order for other configuration data sets may include a step where a search is made for an MVS data set with a specific high-level qualifier: *datasetprefix*. The value of datasetprefix comes from the DATASETPREFIX keyword in the resolver configuration data set or file. If no DATASETPREFIX keyword is found in the resolver configuration data set or file, a default of TCPIP is used by the OpenEdition resolver.

The resolver uses the following search order to locate the actual resolver configuration data set or file to use:

1. The MVS data set or HFS file pointed to by the RESOLVER_CONFIG environment variable

   If the environment variable RESOLVER_CONFIG has been defined, the resolver uses the value of this environment variable as the name of an MVS data set or HFS file to access the resolver configuration data. The syntax for an MVS data set name is "//'mvs.dataset.name'". The syntax for an HFS file name is: "/dir/subdir/file.name".

2. /etc/resolv.conf file

   This file is the preferred place in an OpenEdition system to place the resolver configuration data.

3. Any MVS data set pre-allocated to a DD-name of SYSTCPD

   We discourage anyone from using this technique in an OpenEdition environment because of the restrictions for DD-name allocations during fork() processing (see 1.3.1.1, “Forking a New Process” on page 10).

4. userID.TCPIP.DATA or jobname.TCPIP.DATA

5. SYS1.TCPPARMS(TCPDATA)

6. TCPIP.TCPIP.DATA

   Please observe that if, during TCP/IP for MVS installation, you ran the EZAPPRFX installation job to zap a default high-level qualifier into numerous TCP/IP for MVS modules, this zap does *not* apply to the OpenEdition resolver. The OpenEdition resolver always uses a high-level qualifier of TCPIP in this last search step for a TCPIP.DATA data set.

1.7.2 Protocol Configuration Data

The OpenEdition resolver uses the following search order for a protocol configuration data set or file:

1. /etc/protocol

2. userID.ETC.PROTO or jobname.ETC.PROTO

3. *datasetprefix*.ETC.PROTO

datasetprefix is the value of the DATASETPREFIX keyword in the resolver configuration data set or file.

1.7.3 Service Configuration Data

The services data set or file contains the relationship between service names (servers) and port numbers in the OpenEdition environment. Many server programs query this configuration data set or file via a getservbyname() call to the resolver function. The resolver accesses this data set or file to find the requested service name and returns the port number to use. When you configure /etc/inetd.conf (see 1.8.4, “The InetD Generic Listener Program” on page 35) you specify service names, such as telnet and exec. InetD uses the getservbyname() call to find out which port numbers are assigned to these two services before it begins processing requests.

The following search order is used to find the services data set or file:

1. /etc/services
2. userID.ETC.SERVICES or jobname.ETC.SERVICES
3. *datasetprefix*.ETC.SERVICES

1.7.4 Hosts

If the OpenEdition system does not use a domain name server to resolve host names into IP addresses, the resolver needs access to local hosts tables it can use to resolve host names into IP addresses and IP addresses into host names.

In a UNIX system this is normally accomplished via a flat text file called hosts in the /etc directory: /etc/hosts; the syntax used in this file is generally referred to as BSD-formatted.

In a TCP/IP for MVS system, this is normally accomplished via two data sets, HOSTS.SITEINFO and HOSTS.ADDRINFO, which are built from a flat text data set called HOSTS with a utility program called MAKESITE.

The OpenEdition resolver may use both techniques. The search order for the local host tables is:

1. The MVS data sets pointed to by the X_SITE and X_ADDR environment variables

   If the environment variables X_SITE and X_ADDR have been defined, their values will be used as reference to two fully qualified MVS data sets containing the output from the TCP/IP for MVS MAKESITE utility program, HOSTS.SITEINFO and HOSTS.ADDRINFO. If these two environment variables have been defined, they must point to MVS data sets that are output from the MAKESITE utility. If the environment variables are defined, but the resolver cannot open the data sets, the resolver assumes that you want to use the TCP/IP for MVS HOSTS.ADDRINFO and HOSTS.SITEINFO approach and skips step 2 in the search order, going directly from 1 one to step 3.
2. /etc/hosts
3. userID.HOSTS.xxxxINFO or jobname.HOSTS.xxxxINFO
4. *datasetprefix*.HOSTS.xxxxINFO

1.7.5 ASCII-EBCDIC Translation Table

The resolver has to translate, for example, an EBCDIC host name into an ASCII host name before it sends a request to a name server. When it receives the response from the name server, it has to translate the response from ASCII to EBCDIC before handing over the result to the OpenEdition application program. To perform these translations, the OpenEdition resolver uses the TCP/IP for MVS translation table format, and it searches for a translation table data set using the following search order:

1. The MVS data set or HFS file pointed to by the X_XLATE environment variable

   If the environment variable X_XLATE has been defined it must refer to either a fully qualified MVS data set name or an HFS file name that has been built with the TCP/IP for MVS CONVXLAT utility program.

2. *datasetprefix*.STANDARD.TCPXLBIN

3. An internal default ASCII-EBCDIC translate table

   If the OpenEdition resolver does not find a translate table file or data set, it uses an internal default ASCII-EBCDIC translate table.

1.7.6 Gethostid and Gethostname Calls

In general, the resolver code handles all the get-type calls. There are two calls that require a small comment in this context and they are the gethostid() and gethostname() calls.

1.7.6.1 The Gethostid Resolver Call

The gethostid() call returns the default HOME IP address of this TCP/IP host. This information does not exist in any of the resolver configuration data sets, so the resolver passes this call down to the AF_INET transport provider to obtain the default HOME IP address of the stack. If you use the integrated sockets PFS there is only one stack to pass the request to. If you use the converged sockets PFS, this call is passed to the stack that is listed with the DEFAULT keyword in your BPXPRMxx parmlib member (or the first subfilesystem listed, if you did not specify any DEFAULT keyword).

1.7.6.2 The Gethostname Resolver Call

The gethostname() call returns the host name of this TCP/IP host. Currently this call is passed down to either the one stack that is used with integrated sockets or the default stack if you use converged sockets. If the default stack is a TCP/IP for MVS stack, the host name that is returned is the host name from that stack′s TCPIP.DATA configuration data set and *not* the host name of your OpenEdition resolver configuration data set or file. As this may change in the future, we recommend that you use the same host name in your OpenEdition resolver configuration data set or file as you use in your default stack′s TCPIP.DATA configuration data set. If your default stack is AnyNet MVS, the host name that is returned is the host name from the AnyNet MVS environment data set listed with the keyword HOSTNAME.

1.7.7 Where to Place the Resolver Configuration Data

In general we recommend that you do not share resolver configuration data among the various resolver components, but that you create a separate set of configuration data sets or files per resolver you plan to use.

It is, however, possible to share configuration data sets between, for example, the TCP/IP for MVS resolver and the OE resolver, but you have to be aware of the following limitations for non-OpenEdition applications′ access to HFS files:

- To access a file in the Hierarchical File System, an address space must execute with a user ID that has an OpenEdition UID and GID.
- The application program must access the HFS file using OpenEdition services. An HFS file cannot be accessed via normal QSAM access routines.

None of the TCP/IP for MVS programs that you would normally use, such as NETSTAT or PING, are currently able to access, for example, TCPIP.DATA in the OpenEdition Hierarchical File System. Please observe that it is not enough to do an explicit allocation to an HFS file via the PATH JCL keyword; the application still needs to use OpenEdition services to read the HFS file. If you want to be able to use NETSTAT or any of the TCP/IP for MVS client applications from your TSO session for your OpenEdition TCP/IP for MVS stack, you have to create a TCPIP.DATA MVS data set.

For detailed instructions for creating the OE resolver configuration, please see Chapter 3, “Setting Up the OE Resolver” on page 69.

1.8 Server Programs in OpenEdition

During installation and customization of various server programs in the OpenEdition environment, you have to decide on a number of issues that are related to the way these server programs have been developed and are supposed to run in OpenEdition.

1.8.1 Iterative Server Program

A socket server program can be developed so it will process requests from clients one at a time in a serial fashion. The server program will finish one client request before it is able to receive the next client request. Such a server is called an *iterative server*, and works as shown in Figure 16 on page 33.

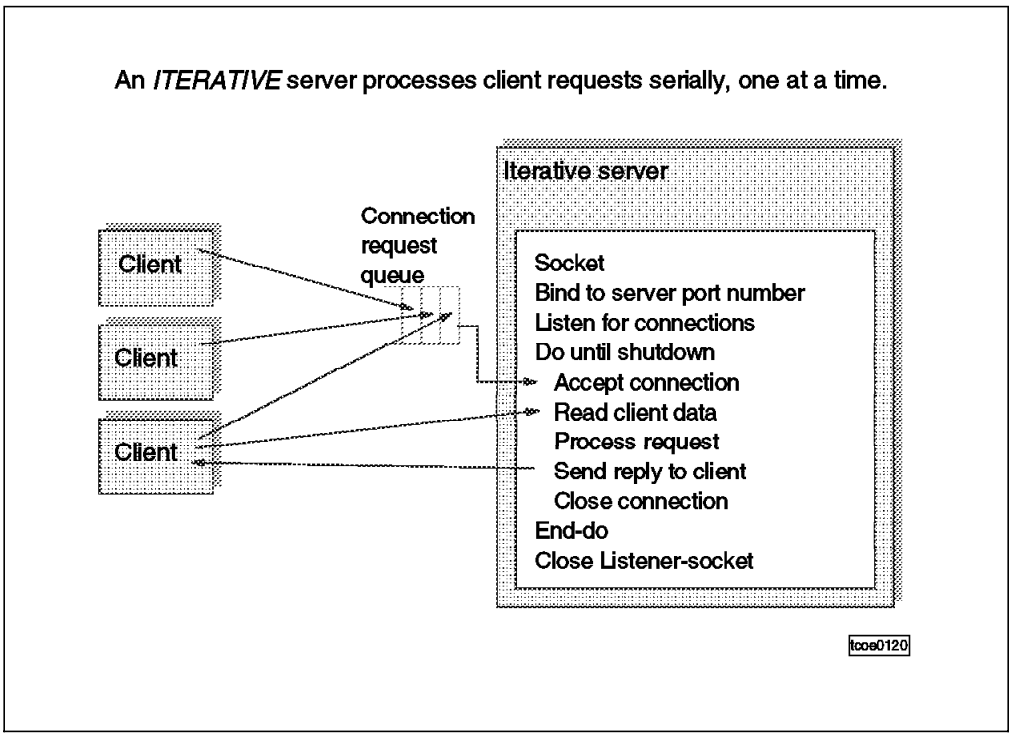


Figure 16. Iterative Server Structure

An iterative server is simple to develop and works well if the number of client requests is small and the processing needed to complete one client request is of a limited duration.

1.8.2 Concurrent Server Program

If the number of client requests is high and/or the time to process individual client requests is of varying length, an iterative server is not an efficient implementation. A *concurrent server* will give a better overall performance. The structure of a concurrent server is shown in Figure 17 on page 34.

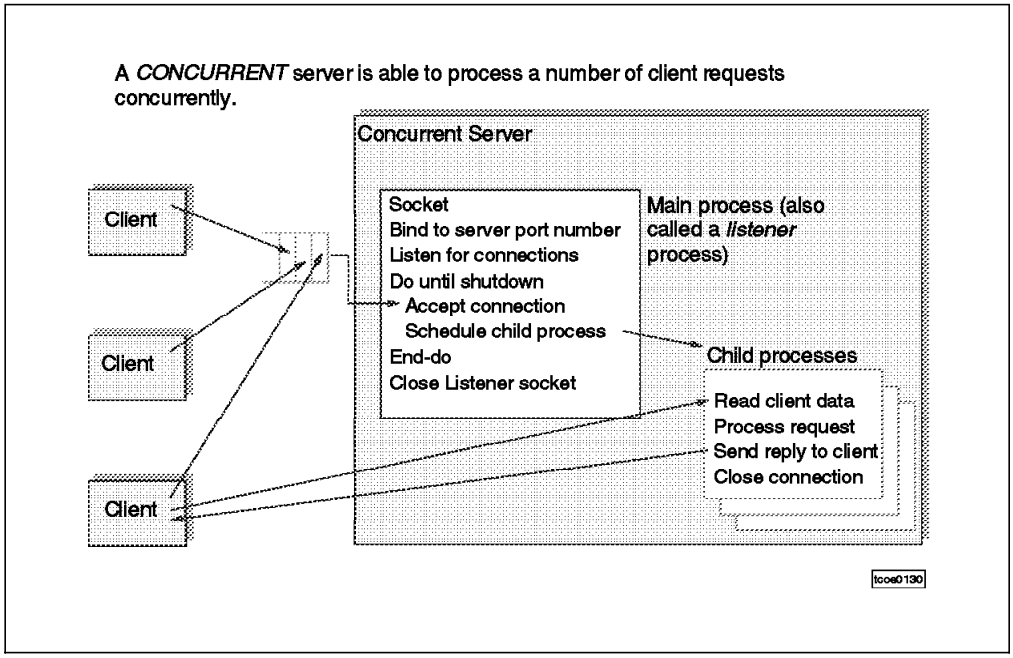


Figure 17. Concurrent Server Structure

A concurrent server consists of two programs:

1. A *listener* program

   This program is actually a small iterative server program, but instead of processing each client request itself, the listener program schedules a child program for every request it receives, and immediately prepares to receive the next client request.

2. A *child* program

   This is the program that is scheduled by the listener program when the listener receives a client request. Each instance of the child program processes one client request and then terminates.

The technique used by concurrent servers allows a high degree of parallel processing, where a number of child programs can execute concurrently, each instance serving one client.

1.8.3 The Listener Program

The listener program may use more techniques in an OpenEdition system to schedule the child programs:

- The listener program may start the child program as a new process via a fork() call followed by an exec() call, or via a single spawn() call. This technique is often used where the individual child processes are supposed to execute for a longer period of time entering a dialog with the client program. An example is an FTP server child program. The FTP client user may enter a series of FTP client requests in one session before the session is terminated on request from the FTP client end user. Because socket descriptors are inherited by the child process, the socket that represents the connected client program is readily available to be used in the child process.
- The listener program may start the child program as a POSIX thread via the pthread_create() function. If the client requests are of short duration, this technique is more efficient than it would be to schedule new processes per

client request. An example of this technique is the Internet Connection Server for MVS. This server schedules child programs to execute as threads within the same process as the listener program is running in. In general a connection only lasts while a single document is being fetched from the Internet Connection Server for MVS. If more documents are required by the Web browser, a series of short connections are used.

1.8.4 The InetD Generic Listener Program

Listener programs are very much alike. The real difference between servers may be seen in the child programs, because it is here the actual application-specific tasks are being performed. In fact the listener program has been generalized to an extent that many different server applications share one and the same listener program. This generic listener program is called *InetD* and is being used by such servers as TelnetD, REXECD, RSHD, just to mention some of those you will meet in this document. Other servers have implemented their own listener programs and cannot use InetD. Of the servers discussed in this document, the FTP server and the Internet Connection Server for MVS do not use InetD, but supply their own listener programs.

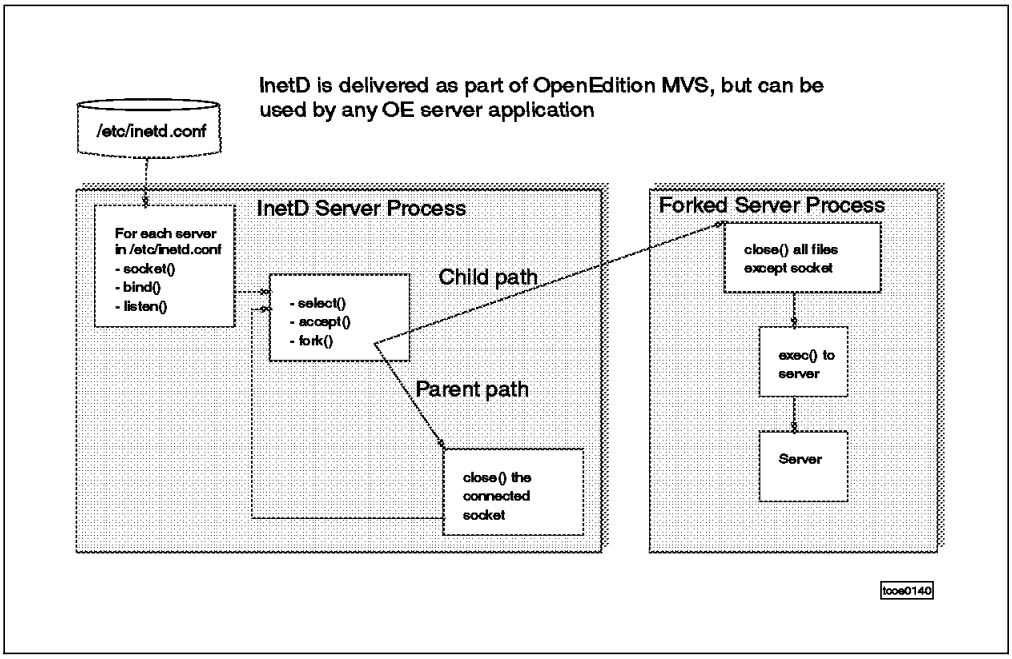


Figure 18. InetD Overview

You control for which server applications InetD is supposed to act as listener. You do so by updating the InetD configuration file, which by default is located in /etc/inetd.conf.

In /etc/inetd.conf you specify the names of the services that InetD supports in your environment. The service names must exist in your /etc/services file, because InetD uses the getservbyname() call to find out which protocol (TCP or UDP) and port number is assigned to the server application. InetD opens sockets, binds them to the port numbers in question, and enters a loop to accept new client requests. When a client request arrives, InetD uses the fork() function to start a new process, where the forked InetD program uses the exec() function to start the server-specific program. The name of this program is specified in the inetd.conf file together with other information for the individual servers.

If you have to pass any run-time options to the server programs that are started via InetD, you have to specify these options in inetd.conf along with the server program name. If you, for example, want to enable tracing in the TelnetD server, you can specify the telnet service line in inetd.conf as follows, where the -D flag is the telnet debug flag and the -t flag is the telnet trace flag:

```
telnet   stream tcp nowait OMVSKERN /usr/sbin/otelnetd otelnetd -l -m -D all -t
```

You need to start your listener programs during OpenEdition startup. In the scenario we describe in this book, the listener programs are:

- InetD - the generic listener for a range of servers
- FTPD - the FTP listener program
- IMWEBSRV - the Internet Connection Server for MVS program

1.8.5 Starting Listener Programs

If no AF_INET transport provider is connected to OpenEdition when you start your listener programs, they will get an error when they try to bind their listener socket to a port number. Some listener programs, such as InetD, handle this situation by writing out a message to syslog:

```
EDC5112I Resources temporarily unavailable
```

The servers then wait a little while before they retry the failed bind() socket call. If an AF_INET transport provider had connected in the meantime, the bind() call is now successful and processing continues normally. Other listener programs, such as FTPD, stop processing in this situation and you have to manually restart the FTPD listener process a little later, when your AF_INET transport provider has connected to OpenEdition.

Depending on how your listener programs handle the above-described situation, you may use different techniques to start them:

- If the listener program includes retry logic, you can start your listener program during OpenEdition initialization from the /etc/rc shell script that is executed during OpenEdition initialization.
- If the listener program does not include retry logic, it may be a better technique to postpone starting the program until after your AF_INET transport provider has connected. Then, either start it manually or develop some automation logic to start the listener program after having received a message that a stack has connected to OpenEdition.

See Figure 19 on page 37 for an overview of the OMVS initialization process.

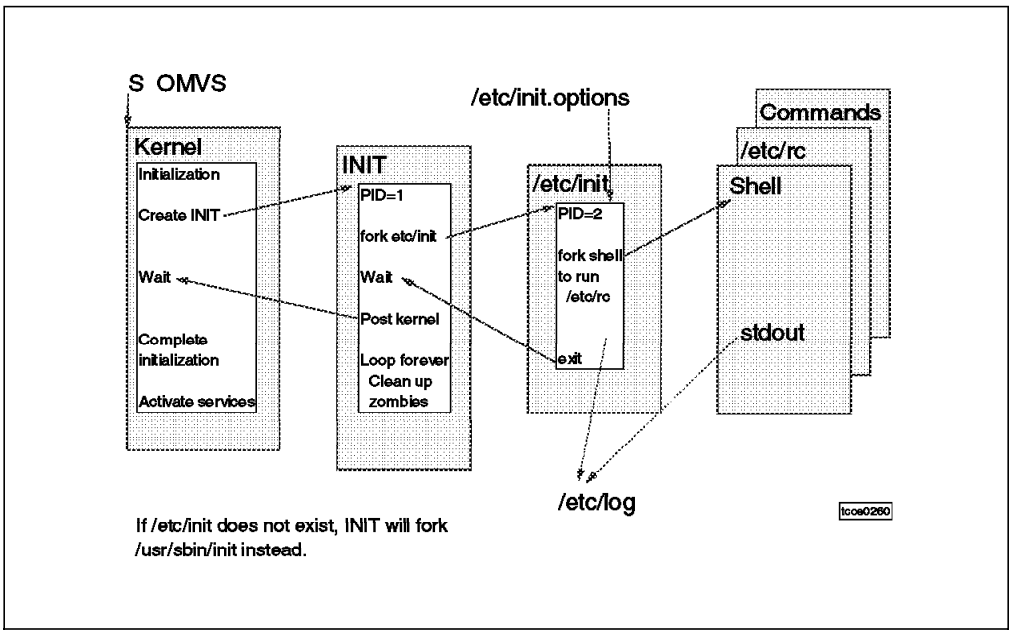


Figure 19. OpenEdition Initialization Overview

1.8.6 When to Recycle Listener Programs

If you use converged sockets and connect, for example, a second AF_INET transport provider some time after having started your OpenEdition environment, you need to recycle your listener processes in order for these to pick up the new AF_INET transport provider. The listen() socket call, which is executed during listener program startup, establishes this listener program as a listener process that will accept client requests, which arrive over the stacks that are connected to OpenEdition at the time the listen() call is issued. When you later connect another stack to your OpenEdition system, the scope of the listen() call that was issued earlier is *not* dynamically extended to the new stack, which means that clients can not connect through the new stack until the listener program has re-issued its listen() call. You have to stop the listener program and restart it in order to do so. Any child processes that were started from the listener process are not affected by a re-cycle of the listener program. Any FTP sessions or telnet sessions that are currently active are *not* affected by a stop and restart of the associated listener process.

1.8.7 Synchronized Port Reservations

When you use converged sockets and connect more TCP/IP for MVS stacks to your OpenEdition environment, you must reserve the same port numbers to OMVS in all stacks that connect to OpenEdition. If you, for example, reserve port number 20 and 21 for OMVS in one stack (to be used by the OE FTPD server) and the same port numbers in the other stack to the non-OE FTP server, the OE-FTP server will not be able to initialize. When the Common I-Net PFS receives the bind() call from the OE FTPD server to bind its socket to port number 20, this call is propagated to both connected TCP/IP for MVS stacks. The stack that did not have port 20 reserved for OMVS will return an error code to the Common I-Net PFS, and the Common I-Net PFS will pass this error code up to the OE FTPD server. The result will be an error message from the OE FTPD server:

```
EDC8115I Address already in use
```

AnyNet MVS currently does not allow you to specify any port reservations, so you do not need to coordinate port reservations between TCP/IP for MVS and AnyNet MVS.

If you have two TCP/IP for MVS stacks and only one of them connects to OpenEdition, you are of course free to reserve port numbers differently in the two TCP/IP for MVS stacks.

1.8.8 Common I-Net Port Range Reservation

For server programs that reside on known port numbers, you reserve these ports in advance via the PORT statements in your TCPIP.PROFILE data sets. These ports are reserved to OMVS, and non-OE programs cannot bind their sockets to ports that are reserved for OMVS.

For client programs, you normally do not assign predefined port numbers, but let the transport protocol layer assign a so-called *ephemeral* (short-lived) port number. Such a port number is assigned out of the range of port numbers above 1024. This assignment takes place at the time the client program calls the connect() function to establish a TCP connection to a remote server, or when a UDP applications calls the sendto() function to send a UDP datagram. The connect() call specifies the IP address of the server host, and the Common I-Net pre-router routes the call to the AF_INET transport provider that offers the best route to the destination IP address. All succeeding socket calls from the client program go to that one transport provider and only that one transport provider has a requirement to manage the ephemeral port number that is used for this client program socket.

Consider an ONC/RPC server program that starts up in an OpenEdition environment and offers its services on all interfaces of all connected AF_INET transport providers, as shown in Figure 20.

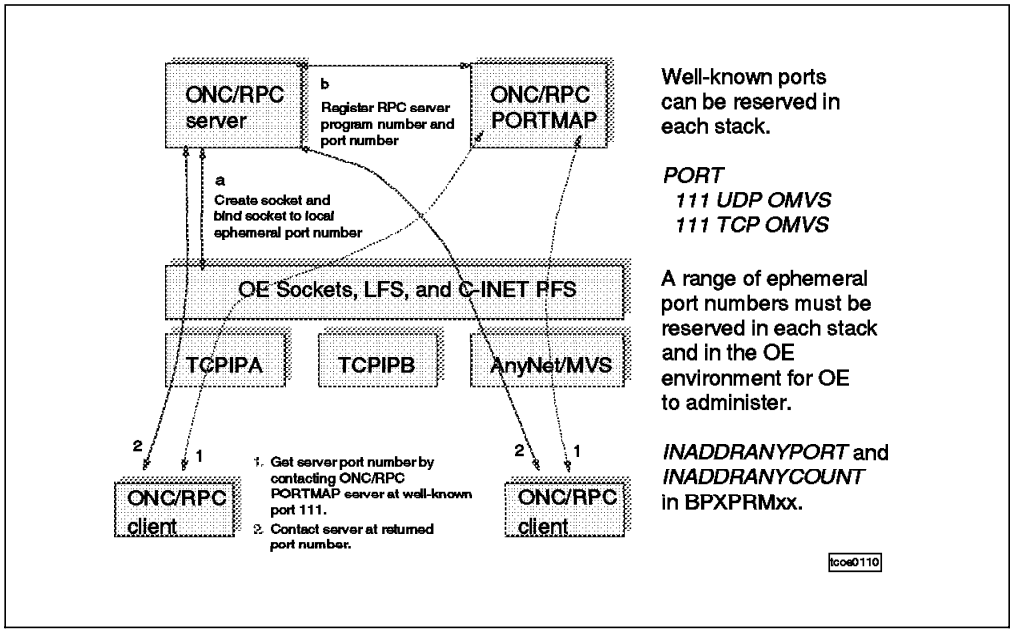


Figure 20. Common I-Net Port Range Reservation

An ONC/RPC server program does not bind its socket to a predefined port number, but lets the transport protocol layer assign an ephemeral port number by means of specifying a port number of zero on its bind() call. When the

ONC/RPC server has bound its socket and retrieved the assigned port number, it registers its ONC/RPC server procedure identity with the local ONC/RPC port mapper that executes on the well-known port 111. The port mapper saves information about the ONC/RPC server identity along with the port number on which it now resides.

ONC/RPC clients in the connected IP networks do not know in advance which port number the individual ONC/RPC server programs reside on, so they start by sending a query to the port mapper that resides on port number 111. The port mapper then returns the port number of the requested ONC/RPC server program and the ONC/RPC client then sends its request to the ONC/RPC server program port number. If you restart the ONC/RPC server program in OpenEdition, it may end up with a different port number from the one it used before you restarted it.

The issue you face in the above scenario is that the port mapper will return the same port number information no matter which of the connected AF_INET transport providers the request was received over. The port number chosen for the ONC/RPC server program must therefore be the same on all connected AF_INET transport providers. If you have a TCP/IP for MVS stack as one of your AF_INET transport providers and you use that stack for both OpenEdition applications and non-OE socket applications, that stack's transport protocol layer has to assign ephemeral port numbers not only to OpenEdition applications, but also to non-OE applications. The ephemeral port numbers that are assigned to OpenEdition applications must be assigned on all connected AF_INET transport providers, but the port numbers that are assigned to local non-OE applications are only known to that one stack. In order to manage this in a consistent and efficient way, each connected AF_INET transport provider must set aside a range of ephemeral port numbers that the stack will *not* assign to its non-OE applications. The specified range is reserved for use by OpenEdition and the assignment of an ephemeral port number to an OpenEdition application takes place in Common I-Net without the need for each connected stack to validate the assignment. Only Common I-Net is allowed to assign port numbers in the specified range, and the stacks must consider the range of port numbers as permanently assigned to OpenEdition.

When you configure your OpenEdition environment, you have to specify the chosen range of port numbers and you have to specify the same range, as follows:

- TCP/IP for MVS PROFILE.TCPIP data set:

```
PORTRANGE 4000 1000 TCP OMVS ; Reserved for OMVS
PORTRANGE 4000 1000 UDP OMVS ; Reserved for OMVS
```

- AnyNet MVS ENVVAR parameter data set:

```
OE_INADDRANY_PORT=4000
OE_INADDRANY_COUNT=1000
```

- BPXPRMxx parmlib member:

```
NETWORK DOMAINNAME(AF_INET)
        ...
        INADDRANYPORT(4000)
        INADDRANYCOUNT(1000)
```

You need to specify a reserved port range even if you do not use ONC/RPC programs in your OpenEdition environment. The reserved port number range is

used in general for assignment of ephemeral port numbers to OpenEdition applications, including any client program that starts in OpenEdition.

1.8.9 Listener Program Overview

See Table 3 for an overview of the listener programs that are used for the servers in the OpenEdition components that are covered in this document.

Table 3. Listener Program Overview

Server function	Listener program	Server child program	Child program environment
Remote login	InetD	/usr/sbin/rlogind	Separate process
Telnet	InetD	/usr/sbin/otelnetd	Separate process
Remote shell	InetD	/usr/sbin/rshd	Separate process
Remote execution	InetD	/usr/sbin/rexecd	Separate process
Echo, Discard, Chargen, Daytime and Time	InetD	Served internally in InetD	
FTP	FTPD	/usr/sbin/ftpdsrv	Separate process
Web Server	IMWHTTPD	Served internally in IMWHTTPD	Separate thread

See Chapter 5, “TCP/IP for MVS OpenEdition Applications Feature” on page 129 for detailed instructions on how to configure and use the servers that are supplied with the TCP/IP for MVS OpenEdition Applications Feature.

1.9 Environment Variables

Environment variables are named variables with assigned values that can be accessed by all programs within an OpenEdition process.

A typical use of environment variables is to initialize a set of environment variables during program initialization so the program can query the value of these variables, and based on the specific values, determine some of the characteristics of the environment in which it is running.

As some of the OpenEdition resolver configuration options can be customized by environment variable settings, we will give you an overview of how and where environment variables can be initialized.

In general, you have to consider two different environments:

- The shell environment
- The POSIX(ON) environment

  Note: In MVS/ESA SP 5.2.2, and in OS/390 POSIX(ON) actually gives you more than a POSIX environment. This level of MVS already provides 90% of what is defined in SPEC1170 or XPG 4.2. So when you use POSIX(ON) you get access to that level of functionality.

1.9.1 Initializing Environment Variables in the Shell

When a login shell is created, either because a user telnets into your OpenEdition system or because a TSO user enters the OMVS command, a number of environment variables are initialized. These variables are initialized based on specifications in the following shell script files:

1. /etc/profile

   This shell script is executed for all users that start a login shell. It holds systemwide environment variables that are to be set up for all users.

   The /etc/profile shell script typically defines variables such as the following:

   ```
   TZ=EST5EDT
   PATH=/bin:.
   NLSPATH=/usr/lib/nls/msg/%L/%N
   LANG=C
   MAIL=/usr/mail/$LOGNAME
   ```

2. $HOME/.profile

   This is an optional shell script that each user can decide to create and place in the user's own home directory. If this script file exists when the user logs in to the shell, it will be used to establish user-specific environment variable settings, including overrides of those variable settings that were established from the systemwide /etc/profile script. It may, for example, set variables depending on nationalities, such as the time zone setting.

   The environment variable TZ may be set in such a way that the user's environment (the *locale*) is automatically adjusted to the time zone the user works in:

   ```
   export TZ='MEZ-1MESZ,086,268'
             1    2    3    4
   ```

   The structure of the TZ environment variable is as follows:

   - In position **1** the name of the standard time zone and its offset to UTC is defined. A negative offset is east of Greenwich, a positive is west of Greenwich. In our case we define the Central European Standard Time. Sometimes you will find it named *CET*.
   - The following four characters (**2**) define the name of the daylight savings time zone. In our case this is MESZ.
   - The last two numbers (**3** , **4**) define the start and end date of the daylight savings period. The dates are defined in form of the 3-digit day number of the year.

     Another way, which may be more convenient, is to define the start and end day in terms of the days of the month. The following example shows the general layout:

     ```
     set TZ='ZZZofsetDDD,Mm.w.d'
     ```

     ZZZofsetDDD defines the name of the zone and the offset relative to UTC as in our previous example.

     Mm.w.d defines the day ($0 \leq d \leq 6$) of week w ($1 \leq w \leq 5$) of month m ($1 \leq m \leq 12$) of the year. Week 5 has the last day (d) in month (m), which may occur in either the fourth or fifth week. Week 1 is the first week in which the dth day occurs. Sunday is day zero.

A setting of MEZ-1MESZ,M3.5.0,M10.5.0 defines that the daylight savings period lasts from the last Sunday in March until the last Sunday in October.

```
Hello HDM.
This is /u/hdm/.profile running
Thu May 16 10:03:39 EDT 1996    1
.
profile executed at Thu May 16 16:03:39 MESZ 1996  2
MEZ-1MESZ,M3.5.0,M10.5.0
/u/hdm/.setup script executed, uid=14(HDM) gid=1(OMVSGRP)
you have mail in /usr/mail/HDM.
pwd=/u/hdm: >
```

Figure 21. Timezone Setting

Setting the TZ variable this way achieves the same effect as you can see in Figure 21. At **1** you see the initial settings and at point **2** the value has been changed according to German time standards.

If an environment variable called ENV has been set in one of the two previous scripts, the value of this environment variable is assumed to be the name of a file that holds additional environment variable definitions for this user. The recommended technique is to set ENV=$HOME/.setup and create a script file in the user's home directory of that name. You can set and export the ENV environment variable in either the /etc/profile or the $HOME/.profile script file. We want to emphasize that it is not a requirement that the ENV variable be defined at all. If it is defined, it is not required to point to $HOME/.setup; you may point to a shell script file of any name. The $HOME/.setup name is just a convention.

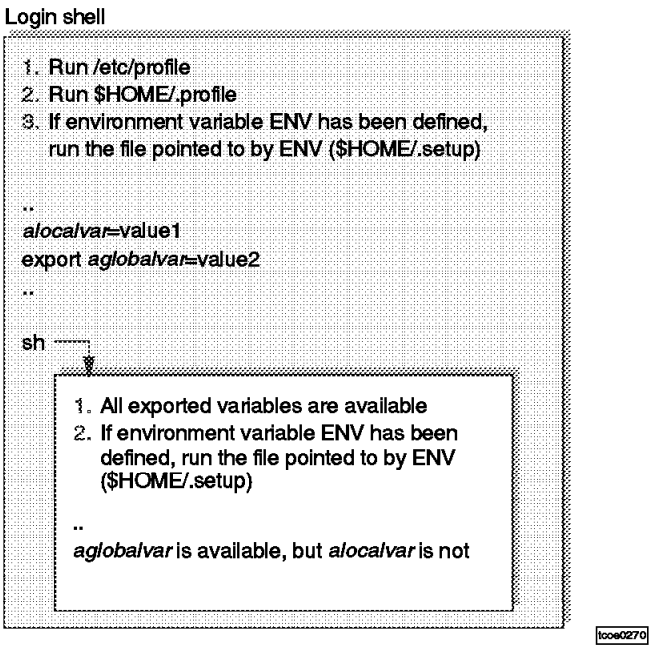


Figure 22. Environment Variables in a Shell Environment

What is the difference between defining your environment variables in $HOME/.profile and in $HOME/.setup? The $HOME/.profile script is executed when you log in to the shell or when you start a new shell with a -L option: sh -L, but not if you create a sub-shell environment, such as running a shell script or using the address SH instruction in a REXX program. The script file specified in the ENV environment variable is executed every time you start a new shell environment.

1.9.2 Environment Variables in the Non-Shell Environment

When you start an OpenEdition program outside the shell environment, none of the environment variables that are set through the techniques described in 1.9.1, “Initializing Environment Variables in the Shell” on page 41 are available to the program.

In order to initialize environment variables for such programs, you have to use one of the following techniques:

- Customize the Language Environment user exits CEEBXITA or CEEBINT. These are initialization exit routines that are invoked during program initialization. The exit routines can define and initialize environment variables.
- Pass environment variables to the program through an EXEC PARM field:

```
// EXEC PGM=MYOEPGM,
//      PARM='POSIX(ON) ENVAR("myvariable=myvalue")/ myparm_1 myparm_2'
```

  Note: Do not leave out the slash in front of your own parameters. This slash separates Language Environment parameters from your parameters passed to MYOEPGM.

- Pass a special environment variable in the EXEC PARM field that identifies a file or data set with additional environment variable definitions:

```
// EXEC PGM=MYOEPGM,PARM='POSIX(ON) ENVAR("_CEE_ENVFILE=DD:MYVARS")/'
//MYVARS  DD *
myvariable=myvalue
mysecondvariable=morevalue
/*
```

  The environment variable _CEE_ENVFILE may point either to an MVS data set, an HFS file, or a DD-name.

1.9.3 Environment Variables and the OpenEdition Resolver

If you decide to use environment variables for the OpenEdition resolver configuration files or data sets, the environment variables are:

1. RESOLVER_CONFIG - the resolver configuration data set or file.
2. X_SITE and X_ADDR - the HOSTS.SITEINFO and HOSTS.ADDRINFO data sets or files.
3. X_XLATE - the ASCII-EBCDIC translate table data set or file built by the TCP/IP for MVS CONVXLAT utility.

If the above environment variables have been defined, they take precedence over any other alternatives for locating the resolver configuration data, which suggests the following scheme:

1. Define the systemwide resolver configuration data sets or files so the OpenEdition resolver, by default, will locate them without using environment variables, for example:
   - /etc/resolv.conf
   - /etc/protocol
   - /etc/services
   - /etc/hosts
   - *datasetprefix*.STANDARD.TCPXLBIN
2. If a program or a user has the need to override the systemwide resolver configuration data, the environment variables can be used on an individual basis as described in 1.9.1, “Initializing Environment Variables in the Shell” on page 41 and 1.9.2, “Environment Variables in the Non-Shell Environment” on page 43.

See Chapter 3, “Setting Up the OE Resolver” on page 69 for more details of how to configure the OE resolver.

1.10 AnyNet and OpenEdition

AnyNet is a family of software products consisting of multiprotocol access nodes and multiprotocol gateway nodes that are based on the multiprotocol transport networking (MPTN) architecture.

The multiprotocol transport networking architecture, announced in March 1993, describes the logical structures, formats, protocols and operating principles that allow applications to use networks other than the one originally written for, without any change to the existing application.

Traditionally, networking APIs are tied to one particular network protocol family. For example, if you develop a program that uses the sockets API, such a program is traditionally tied to the TCP/IP protocol stack. If you develop a program that uses the CPI-C API, such a program is traditionally tied to the SNA protocol stack. Multiprotocol transport networking removes the tie between a particular API and a particular network protocol family, allowing your socket programs to use an SNA network and your CPI-C programs to use a TCP/IP network. Other APIs and networking protocol families are supported too, but in our context in this book the focus is on the sockets over SNA functions.

See Figure 23 on page 45 for an example of AnyNet node types.

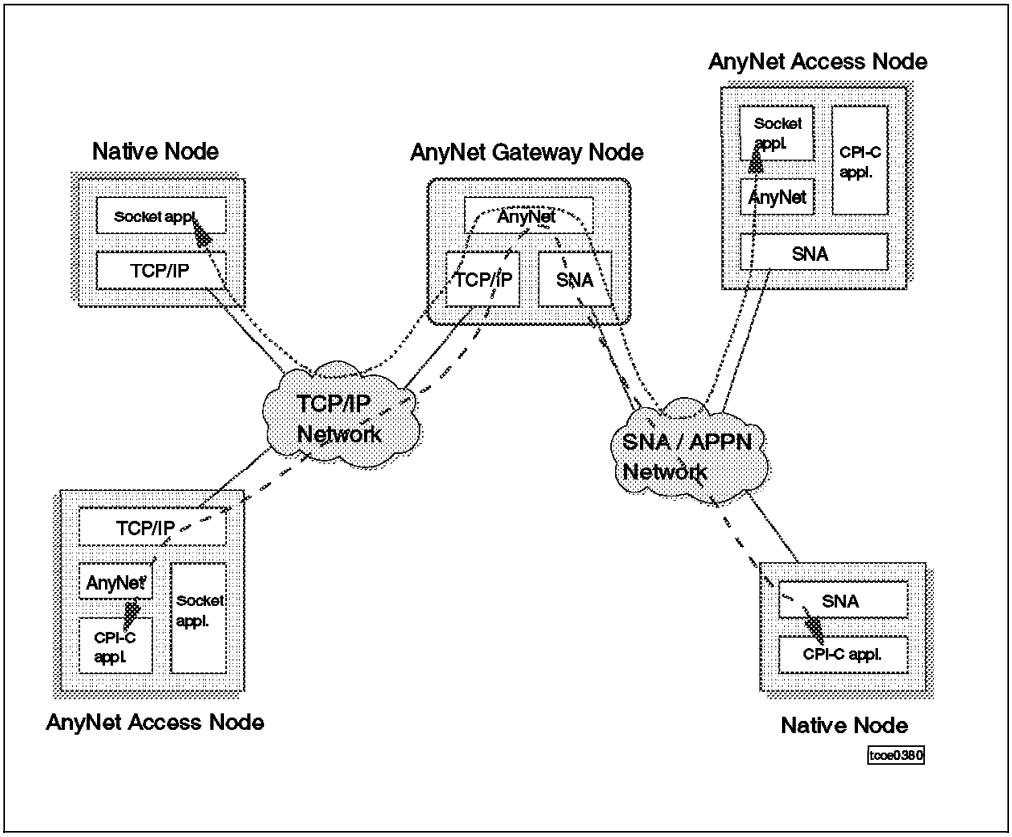


Figure 23. AnyNet Node Types

AnyNet implements two different node types:

- An AnyNet Access Node

  An access node provides functions that allow an application program running on the access node to use a network for which the API was not originally intended. On an access node you can, for example, run a socket program that uses an SNA network to communicate with another socket program on another access node.

- An AnyNet Gateway Node

  A gateway node connects two different networks and provides network protocol conversions between the related network protocols. A gateway node can, for example, connect a TCP/IP and a SNA/APPN network, allowing a socket program on an access node in the SNA/APPN network to communicate with a socket program that runs on a native TCP/IP host attached to the TCP/IP network.

A native node is defined in the MPTN architecture as a node that does not implement MPTN functions. An example could be a non-IBM UNIX platform that implements native socket applications on a standard TCP/IP stack. To use AnyNet functions, such a host has to route IP packets via an AnyNet gateway node.

When the application and the networking protocol layer use different addressing schemes, AnyNet performs address mapping between them. Address mapping is based on one of the following methods:

- Algorithmic, used for Sockets over SNA

- Extended protocol-specific directory, used for SNA over TCP/IP
- Address mapper, used for NetBIOS over SNA

When you use the Sockets over SNA functions of AnyNet, the involved AnyNet nodes have to map IP addresses to SNA logical unit names.

If you look at a combined TCP/IP and AnyNet network from an IP network topology point of view, the SNA/APPN section of the network constitutes a separate IP net or subnet. The Sockets over SNA gateway nodes act as normal IP routers that route IP packets between two interfaces: one standard TCP/IP interface and an AnyNet interface (the *sna0*) interface. See Figure 24 for an overview of the IP network topology in a combined TCP/IP and AnyNet network.

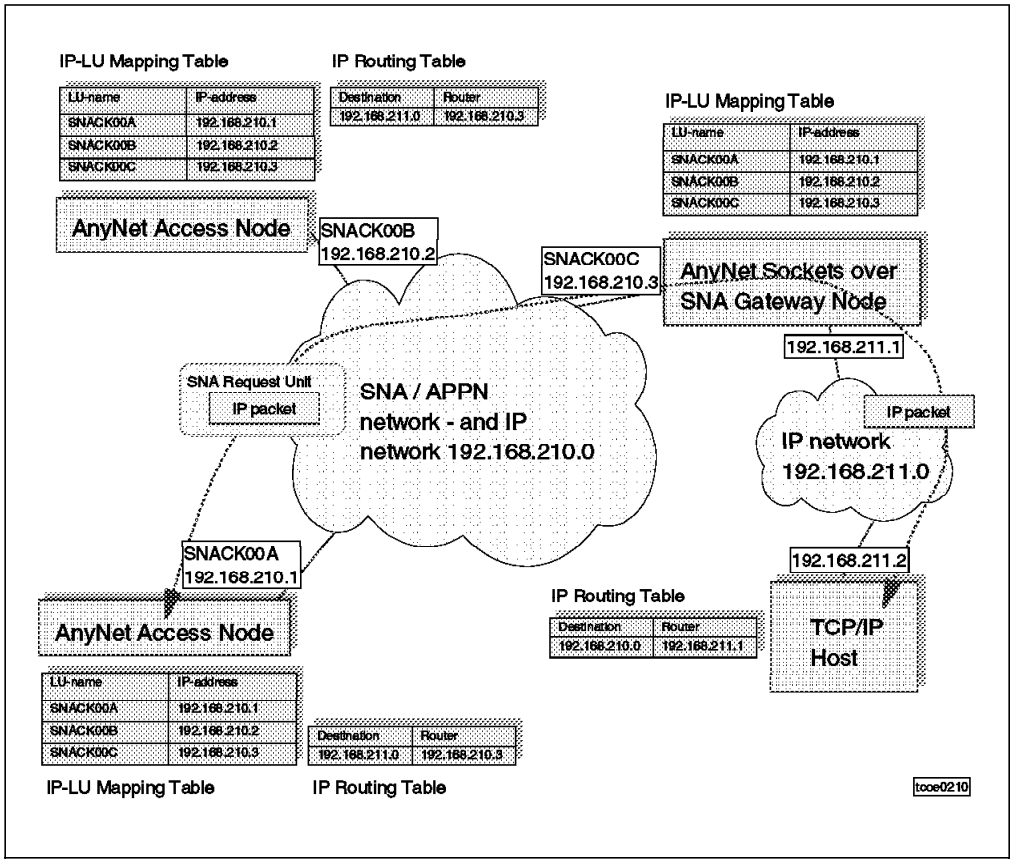


Figure 24. IP Topology of a Combined TCP/IP and AnyNet Network

AnyNet products are available on the following platforms:

- MVS/ESA
- OS/2
- AIX/6000
- OS/400
- Windows
- 2217 Multiprotocol Converter

For an overview of AnyNet access node implementations, please see Table 4 on page 47.

Table 4. AnyNet Access Nodes						
access nodes	**MVS**	**OS/2**	**AIX**	**OS/400**	**WIN**	**2217**
Sockets over IPX		yes		yes		
SNA over IPX				yes		
Sockets over NetBIOS		yes				
NetBIOS over SNA		yes				
Sockets over SNA	yes	yes	yes	yes		
SNA/APPC over TCP/IP	yes	yes	yes	yes	yes	

For an overview of AnyNet gateway node implementations, please see Table 5.

Table 5. AnyNet MVS Gateway Nodes						
Gateway Nodes	**MVS**	**OS/2**	**AIX**	**OS/400**	**WIN**	**2217**
IPX over TCP/IP		soon				
IPX over SNA		yes				yes
NetBIOS over TCP/IP						
NetBIOS over SNA		yes				yes
Sockets over SNA		yes	yes			yes
SNA over TCP/IP	yes	yes	yes			yes

1.10.1 AnyNet MVS

In an MVS environment, AnyNet functions are available via MPTF (Multiple Protocol Transport Facility) or via the AnyNet MVS feature of VTAM/ESA. The first VTAM/ESA version to support AnyNet MVS was VTAM/ESA V3.4.2.

In VTAM for MVS/ESA V4.2, APAR OW10895 / PTF UW17057 supplied OpenEdition support for integrated sockets and thus enabled AnyNet MVS as an AF_INET transport provider in an OpenEdition environment.

The support for OpenEdition converged sockets is included in the AnyNet MVS feature of VTAM V4.3 for MVS/ESA.

When you use AnyNet MVS as an AF_INET transport provider for OpenEdition, you configure AnyNet MVS as a Sockets over SNA access node.

The AnyNet MVS feature of VTAM V4.3 for MVS/ESA supplies the following functions:

- Used as an AnyNet Access Node:
  - Sockets over SNA
  - SNA / APPC over TCP/IP for MVS
- Used as an AnyNet Gateway Node:
  - SNA / APPC over TCP/IP for MVS

The Sockets over SNA implementation in AnyNet does not support broadcast IP packets. This means that applications that rely on IP level broadcast functions do not work with AnyNet. Typical examples of such applications are dynamic IP route update servers, such as the RouteD server that is based on the RIP Version 1 protocol. This means that you will have to configure your AnyNet nodes with static IP routing tables.

Please see 4.5, “AnyNet MVS, Single Stack” on page 98 for detailed instructions on how to set up AnyNet MVS as an OpenEdition AF_INET transport provider.

Chapter 2. Security

With the advent of OpenEdition, not only were new philosophies in programming and end-user interaction introduced, but we were also faced with the challenge of how to provide this new environment with a level of security that was very similar to what MVS/ESA had provided for years. The major contributor to the implementation of such a level of security is of course RACF, which provides the new security functions by implementing new general resource profiles and new segments to the existing user and group profiles.

Note: Though we are speaking here about RACF only, other vendors of security products also support the OpenEdition environment.

The purpose of this chapter is to collect all RACF relevant data into one place instead of repeating it throughout the book. We assume that you already know the basics of OpenEdition security and that you already have BPX.SUPERUSER security defined in your system. We restrict ourselves to the type of security that is related to the use of server functions in OpenEdition, and in particular to the effects of the BPX.DAEMON and BPX.SERVER facility classes.

This chapter includes the following topics:

- 2.1, Program Control
- 2.2, BPX.DAEMON Facility Class
- 2.3, Started Task User IDs
- 2.4, OMVS Kernel Address Space
- 2.5, InetD
- 2.6, TCP/IP for MVS
- 2.7, FTPD
- 2.8, Internet Connection Server for MVS
- 2.9, Accounting
- 2.10, Summary

© Copyright IBM Corp. 1996

2.1 Program Control

With the implementation of UNIX services in MVS/ESA, a new location for executable programs is introduced, the Hierarchical File System. In UNIX terminology a program is an *executable file*, or for short, just an *executable*. In standard MVS we work with load modules that reside in load libraries. A search for an MVS load module is performed in a predefined sequence of steps that include already loaded modules, STEPLIB or JOBLIB libraries, LPA resident modules, and modules that reside in libraries that are included in the system link list as specified in the LNKLSTxx member of SYS1.PARMLIB. There are techniques in standard MVS to work with authorized load modules, which are programs that use certain authorized functions in an MVS system, for example, changing the MVS identity of an address space or a task in an address space via the RACROUTE REQUEST=VERIFY function.

As the Hierarchical File System is a UNIX-style file system, OpenEdition uses UNIX-style techniques when searching for executable files in the Hierarchical File System. The order of search is specified via an environment variable called PATH. Because every user may set his or her own PATH variable, it is relatively simple for a user to specify that the user's own programs should execute instead of system programs or commands. The user can, for example, assign the following value to the PATH environment variable:

```
.:/bin:/u/hdm:/usr/lpp/internet/www/Admin
```

1. The leading dot (.) specifies that the search for an executable file should begin in the current directory.
2. If the executable file is not found in the current directory, then the search continues through the `/bin`, `/u/hdm`, and finally `/usr/lpp/internet/www/Admin` directories.

As you can see by the above example, it is quite simple for a user to *fake* a system command or system program through one of his or her own programs. This is especially dangerous when you deal with server programs, or *daemons* as server programs normally are called in a UNIX environment. Such never-ending server programs are often responsible for managing session setup by requesting the user ID and password from users who enter the system using, for example, telnet. By faking InetD you will be able to record the user IDs and passwords of all users that log in via telnet.

These are some of the reasons why MVS considers the entire Hierarchical File System to be an unauthorized library.

Fortunately there are means provided by RACF to prevent such a *trojan horse* behavior. We explain these means in the following sections.

At first we have to look at the *sticky bit*. The sticky bit was invented with early versions of UNIX where the process of loading an executable file into memory was very resource consuming. By setting the sticky bit in the file system for the executable file, the executable file was kept in storage even after it had finished executing, ready to be used by other users in the UNIX system. It was kept in storage until the system was shut down. Nowadays systems are not as constrained in storage as they used to be, so the original meaning of the sticky bit has vanished and it is now often used for other purposes.

OpenEdition uses the sticky bit to bypass loading of an executable from the Hierarchical File System. If an executable file has the sticky bit turned on as shown in Figure 25 on page 51, the executable file in the Hierarchical File System will not be used, but MVS will instead turn to the standard MVS search order to look for a copy of the executable file in an MVS load library.

```
pwd=/usr/lpp/internet: >ls -l
total 40
-rw-r--r--   2 WEBADM   IMWEB         93 Jan 24 01:52 envvars
-rw-r--r--   2 WEBADM   IMWEB      11840 Feb  8 00:59 httpd_msg.cat
drwxr-xr-x   2 WEBADM   IMWEB          0 Jan 24 01:53 IBM
-rwxr--r-T   2 WEBADM   IMWEB        237 Jan 24 01:52 IMWCGIBN
drwxr-xr-x   2 WEBADM   IMWEB          0 Jan 24 01:31 logs
drwxr-xr-x   3 WEBADM   IMWEB          0 Feb 16 22:38 Samples
drwxr-xr-x  10 WEBADM   IMWEB          0 Mar 14 21:36 ServerRoot
pwd=/usr/lpp/internet: >
```

Figure 25. Sticky Bit Setting for an Executable File

The T in the file security packet for the file called IMWCGIBN is the sticky bit.

When the OpenEdition program loader finds an executable file with the sticky bit on, it does not load the program from the Hierarchical File System, but passes the load request on to the MVS contents supervisor, which then searches for the requested module in one of the following libraries:

1. STEPLIB, if it has been allocated to the current process
2. Link libraries as they are defined in the LNKLSTxx member of SYS1.PARMLIB

Task libraries need not be taken into account, as they cannot be defined in any way.

In order to ensure that the load module that is eventually found by the MVS contents supervisor is a *known and trusted* load module, RACF *program control* must be active. The RACF program class profile for a controlled program includes both the name of the module and the library where this module resides, ensuring that only a *known* version of a load module is executed.

When you use program control, you must ensure that all load modules that are loaded into an address space come from controlled libraries. If a module is loaded from a noncontrolled library, the address space becomes *dirty* and loses its authorization. When the MVS contents supervisor loads a module from a noncontrolled library, it sets the *dirty bit*, and the authorization for the entire address space is lost. This means that you must define all the libraries, from where load modules may be loaded, as program controlled. This includes the C run-time library, the TCP/IP for MVS SEZALINK library, the Internet Connection Server for MVS load library, and SYS1.LINKLIB as a minimum.

To activate program control and to identify programs as controlled, use the following RACF commands:

- To activate program control

```
SETROPTS WHEN(PROGRAM)
```

- To prevent unauthorized users from updating the program libraries, you should, if you have not already done so as part of your normal system setup, protect your load libraries via RACF data set profiles:

```
ADDSD 'cee.version.SCEERUN' UACC(READ)
ADDSD 'imw.version.SIMWMOD1' UACC(READ)
ADDSD 'SYS1.LINKLIB' UACC(READ)
ADDSD 'SYS1.TCPIP.SEZALINK' UACC(READ)
```

  The universal access for public library data sets (those in LNKLSTxx), may be set to NONE. Users can still use the controlled programs and any other program in those libraries, because MVS opens the LNKLSTxx libraries during IPL and makes these programs public. By doing this, users are prevented from opening the libraries and copying modules from them.

- Define the load libraries in question as controlled libraries:

```
RDEFINE PROGRAM * ADDMEM('SYS1.LINKLIB'/volser/NOPADCHK) UACC(READ)
RALTER  PROGRAM * ADDMEM('cee.version.SCREERUN'/volser/NOPADCHK) UACC(READ)
RALTER  PROGRAM * ADDMEM('imw.version.SIMWMOD1'/volser/NOPADCHK) UACC(READ)
RALTER  PROGRAM * ADDMEM('SYS1.TCPIP.SEZALINK'/volser/NOPADCHK) UACC(READ)
```

  Note: Do *not* specify a universal access of NONE for the PROGRAM resources. If you do so for your SYS1.LINKLIB programs, you will not be able to IPL your MVS system.

By combining the use of the sticky bit and the RACF program control functions, you can ensure that MVS is using exactly that version of a load module you defined to be the trusted one.

2.2 BPX.DAEMON Facility Class

After having set up RACF definitions to make sure that OpenEdition only loads trusted server programs, we need a way to specify that such a server program is allowed to change the user security environment of the process in which it is executing. This must work for programs for which there is no requirement that they come from an APF authorized library, which means that they do not have a way to bring themselves into an MVS authorized state via use of MODESET macro calls. In other words, the technique must work for programs that execute in KEY=8.

The RACF facility class resource *BPX.DAEMON* is used for this purpose. The user ID under which your server programs are initially started must have READ access to BPX.DAEMON, in order for the server program to change the user security environment to that of another user ID.

The authorization to change the user security environment is only granted if both of the following two conditions are true:

1. The server program is executing under a user ID that has READ permission to the BPX.DAEMON facility class profile.
2. All programs running in the address space have been fetched from a controlled library. The Hierarchical File System is not considered to be a controlled library.

To enable BPX.DAEMON security, you have to define the BPX.DAEMON facility class profile in RACF:

```
RDEFINE FACILITY BPX.DAEMON UACC(NONE)
```

Note: You *must* use the name BPX.DAEMON in this command. Substitutions for the name are not allowed.

As soon as you define the BPX.DAEMON resource, MVS will not let programs change the security environment unless the programs were fetched from a program controlled library and the user ID under which the program executes has access to BPX.DAEMON. So, please plan the point in time where you define BPX.DAEMON carefully. If all the required conditions are not met, your server programs will fail as soon as you define BPX.DAEMON. If they do fail, you may delete BPX.DAEMON again and your setup will revert to what it was before. You can then go back and check all your definitions and make any required corrections before you try to define BPX.DAEMON again.

If this is the first FACILITY class profile that your installation is going to use, you need to activate the FACILITY class with the following commands:

```
SETROPTS CLASSACT(FACILITY) GENERIC(FACILITY) AUDIT(FACILITY)
SETROPTS RACLIST(FACILITY)
```

Depending on the technique you use to start your server programs, some of them may inherit their initial user identity from the kernel address space. This will be the case if you start server programs from the /etc/rc shell script that is executed during startup of OpenEdition. In that case, the user ID of the kernel address space must be authorized to the BPX.DAEMON resource:

```
PERMIT BPX.DAEMON CLASS(FACILITY) ID(kernel_user_ID) ACCESS(READ)
```

If you start server programs via MVS start commands or from shell scripts that execute after startup of OpenEdition, you need to further allow these user IDs access to the BPX.DAEMON facility class resource. This could, for example, be the user IDs under which you start the Internet Connection Server for MVS and the FTPD server:

```
PERMIT BPX.DAEMON CLASS(FACILITY) ID(ftpd_user_ID) ACCESS(READ)
PERMIT BPX.DAEMON CLASS(FACILITY) ID(wwwserver_user_ID) ACCESS(READ)
```

Note: wwwserver_user_ID and ftpd_user_ID are the user IDs you are going to use for these processes.

2.3 Started Task User IDs

With RACF 2.1 a new technique was provided to associate a user ID with a started task, the *STARTED* resource class. Until then this was done through the configuration table ICHRIN03. The major benefit of the new support is that changes can be applied without an IPL. The following example (Figure 26 on page 54) shows how to define a started resource profile for the MVS Web server.

```
RDEFINE STARTED TCPW3SRV.* -
  STDATA(USER(TCPW3SRV) GROUP(TCPW3GRP) PRIVILEGED(NO) TRUSTED(NO) TRACE(NO))
```

Figure 26. Define the STARTED Profile for the Web Server

As the started resources are kept in storage, you need to refresh the started resource class every time you have made changes to the definitions.

When you start the Web server via a START TCPW3SRV MVS operator command, you will see the following messages on the MVS console:

```
IEF695I START TCPW3SRV WITH JOBNAME TCPW3SRV IS ASSIGNED TO USER
TCPW3SRV, GROUP TCPW3GRP
```

2.4 OMVS Kernel Address Space

The OMVS kernel address space is the “owner” of every process it forks on behalf of /etc/rc definitions. That means the user ID of the kernel address space must be defined to BPX.DAEMON with READ access.

In our sample setup, the OMVS user ID was OMVSKERN. If you start InetD and SyslogD via the /etc/rc shell script, they will execute under the OMVSKERN user ID as shown in Figure 27.

```
RESPONSE=SA18
 BPXO001I 10.41.03 DISPLAY OMVS 167
 OMVS     ACTIVE                                                BPXPRM18
 USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
 OMVSKERN OMVSINIT 002F          1          0 MRI   17.27.57       .173
   SERVER=Init Process                     AF=    0 MF=65535 TYPE=FILE
 T18OTCP  T18OTCP  0044     196610          1 MR    17.34.27    169.820
 HDM      HDM      0035     458755     983049 1CI   09.29.10     14.790
   LATCHWAITPID=         0 CMD=sh -L
 CS2201SR CS2201SR 0050     458756          1 HFI   17.35.25      2.252
 HDM      HDM      0048     589829     393223 1CI   08.58.49      4.440
   LATCHWAITPID=         0 CMD=sh -L
 OMVSKERN SYSLOGD4 0028          6          1 1F    17.28.06     10.363
   LATCHWAITPID=         0 CMD=/usr/sbin/syslogd
 HDM      HDM      0048     393223         11 1FI   08.58.42      4.600
   LATCHWAITPID=         0 CMD=rlogind2 -a -m vt220/9600 hdm rs60003.it
 HDM      HDM      0049     327688          1 MR    10.07.21     28.314
 HDM      HDM      0035     983049         11 1FI   09.29.05     18.480
   LATCHWAITPID=         0 CMD=rlogind2 -a -m vt220/9600 hdm rs60003.it
 OMVSKERN INETD8   0030         11          1 1FI   17.35.15       .446
   LATCHWAITPID=         0 CMD=/usr/sbin/inetd /etc/inetd.conf
 ALFREDC  ALFREDC  003E     458765          1 1RI   10.01.42     31.016
 OMVSKERN FTPD1    0015     327696          1 1FI   10.37.53       .287
```

Figure 27. Ownership of Processes Started Through /etc/rc

2.5 InetD

As we described in 1.8.4, “The InetD Generic Listener Program” on page 35, InetD is a generic listener program that is responsible for starting server programs on the user′s request. The operation of InetD is controlled through its configuration file /etc/inetd.conf.

When InetD is running and receives a request for a connection, it processes that request according to the port number over which it received the request. If, for example, a user tries to log in from a remote system using rlogin, InetD receives the request at port number 513. The correlation between port numbers and service names as they are specified in /etc/inetd.conf is established through your /etc/services file, where you define which port number is associated with a service name. After having received the connection request, InetD issues a fork(), and the forked process issues an exec() to start the requested server program (in this case the RloginD program) which then runs as the server.

The user ID under which the new process will initially start is defined in /etc/inetd.conf.

```
#======================================================================
# service | socket | protocol | wait/ | user | server  | server program
# name    | type   |          | nowait|      | program |   arguments
#======================================================================
login      stream    tcp       nowait OMVSKERN /usr/sbin/rlogind rlogind -m
                                      1
```

Figure 28. Definition of a Service in /etc/inetd.conf

In Figure 28 we have extracted the definition line for the rlogin service. The item that we discuss here is the user ID of the server process (1). In this example, we have assigned a user ID of OMVSKERN to this server program. After InetD has created a new process and started the RloginD server program in that process, RloginD will obtain a user ID and password from the rlogin client and change the security environment from OMVSKERN to that of the client user. As a consequence of this behavior of RloginD, the user ID specified in /etc/inetd.conf for the login service must have an OMVS RACF segment defined to it and access to the FACILITY class profile BPX.DAEMON.

In the sample setup we established during creation of this book, the InetD servers were all started with the OMVSKERN user ID, which we had defined with READ access to the FACILITY class profile BPX.DAEMON. If you are using different user IDs for each server, you will have to define these accordingly.

2.5.1 TelnetD, RSHD, REXECD and RloginD

The most often used services that are managed by InetD are telnet, RSH, REXEC, and rlogin. Fortunately, the processing needed by InetD to handle these services are more or less the same, so we can restrict ourselves to describing one scenario in detail, which will be the telnet service.

InetD listens for telnet connection requests on port 23. Immediately after such a request arrives, InetD clones itself by means of the fork() service and starts the TelnetD server program as specified in /etc/inetd.conf.

```
RESPONSE=SA18
 BPXO001I 16.26.06 DISPLAY OMVS 350
 OMVS     ACTIVE                                              BPXPRM18
 USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
 OMVSKERN OMVSINIT 002A          1          0 HRI P 13.16.31       .106
   SERVER=Init Process                     AF=    0 MF=65535 TYPE=FILE
 RAISOCK  RAISOCK  0044     131074          1 MR    13.20.07    461.524
   SERVER=SNACKETS                         AF=    0 MF=65535 TYPE=FILE
 CS2201SR CS2201SR 0051     327684          1 HFI   08.55.25      2.193
 OMVSKERN SYSLOGD4 0023      65541          1 1FI   13.16.41     36.352
   LATCHWAITPID=         0 CMD=/usr/sbin/syslogd
 ALFREDC  INETD7   002E     917512     327696 1CI   16.15.59      1.680
   LATCHWAITPID=         0 CMD=sh -L
 OMVSKERN FTPD1    0021    1245193          1 1FI   13.33.22       .263
 T18OTCP  T18OTCP  004E     196618          1 MR    13.31.53     36.807
 OMVSKERN INETD1   002C    1114123     655373 1FI   16.26.00       .339
   LATCHWAITPID=         0 CMD=otelnetd -l -m -D all -t
 OMVSKERN INETD5   002B     655373          1 1FI   16.15.21       .763
   LATCHWAITPID=         0 CMD=/usr/sbin/inetd /etc/inetd.conf
 HDM      HDM      0040      65550          1 1R    11.38.24     21.280
 ALFREDC  ALFREDC  003E     720911          1 MRI   16.04.39     52.250
 ALFREDC  INETD7   002E     327696     655373 1FI   16.15.47      3.000
   LATCHWAITPID=         0 CMD=otelnetd -Y 192.168.210.8 -p alfredc -a
```

Figure 29. OMVS Status Just after Telnet Connection

The OMVS status display just after a new telnet connection has been processed is shown in Figure 29.

The TelnetD server program enters the initial session setup dialog with the telnet client and requests a user ID and password from the client end-user. In this phase, user ID and password are checked. After these are accepted by the current phase of TelnetD, TelnetD restarts itself via a new exec() call with a new set of parameters and in addition, a shell process is started. See Figure 30 on page 57 for an OMVS status display after the full telnet session has been established and the end-user is ready to enter shell commands.

```
RESPONSE=SA18
 BPXO001I 16.26.40 DISPLAY OMVS 356
 OMVS     ACTIVE                                          BPXPRM18
 USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
 OMVSKERN OMVSINIT 002A          1          0 MRI   13.16.31       .106
   SERVER=Init Process                     AF=    0 MF=65535 TYPE=FILE
 RAISOCK  RAISOCK  0044     131074          1 MR    13.20.07    461.613
   SERVER=SNACKETS                         AF=    0 MF=65535 TYPE=FILE
 HDM      HDM      002C     851971    1114123 1CI   16.26.29       .770
   LATCHWAITPID=         0 CMD=sh -L
 CS2201SR CS2201SR 0051     327684          1 HFI   08.55.25      2.195
 OMVSKERN SYSLOGD4 0023      65541          1 1FI   13.16.41     37.005
   LATCHWAITPID=         0 CMD=/usr/sbin/syslogd
 ALFREDC  INETD7   002E     917512     327696 1CI   16.15.59      1.680
   LATCHWAITPID=         0 CMD=sh -L
 OMVSKERN FTPD1    0021    1245193          1 1FI   13.33.22       .263
 T18OTCP  T18OTCP  004E     196618          1 MR    13.31.53     37.001
 HDM      HDM      002C    1114123     655373 1FI   16.26.00      1.160
   LATCHWAITPID=         0 CMD=otelnetd -Y 9.24.104.208 -p hdm -a vt220
 OMVSKERN INETD5   002B     655373          1 1FI   16.15.21       .763
   LATCHWAITPID=         0 CMD=/usr/sbin/inetd /etc/inetd.conf
 HDM      HDM      0040      65550          1 1R    11.38.24     21.300
 ALFREDC  ALFREDC  003E     720911          1 MRI   16.04.39     52.250
 ALFREDC  INETD7   002E     327696     655373 1FI   16.15.47      3.000
   LATCHWAITPID=         0 CMD=otelnetd -Y 192.168.210.8 -p alfredc -a
```

Figure 30. OMVS Status After TELNET User Identification

Notice that both the TelnetD process and the shell process execute under the end-user user ID, and not the original OMVSKERN user ID. Look at the PPID (parent process ID) column. It shows nicely the relationship between the processes. The shell process has the TelnetD process as parent, the TelnetD process has the InetD process as parent, and InetD in turn has the grand daddy of all processes, the OMVSINIT process, as parent.

The process for the other interactive services, such as RSHD, REXECD, and RloginD, are similar. You will need almost the same definitions and will observe the same behavior as in Figure 29 on page 56 and Figure 30.

2.6 TCP/IP for MVS

The AF_INET physical file system of OpenEdition uses the TCP/IP system address space as an AF_INET transport provider as discussed in 1.6, “Sockets in the OpenEdition Environment” on page 19. To achieve this integration, the TCP/IP system address space must connect to OpenEdition and be dubbed as an OpenEdition process, which is the reason why the started task user ID that is assigned to the TCP/IP system address space must have a valid OMVS segment. As an AF_INET transport provider, the TCP/IP system address space requires superuser privileges in OpenEdition. You can either define the TCP/IP system address space started task user ID with a UID=0, or you can define the TCP/IP system address space as a trusted environment in the RACF started class profile for the TCP/IP system address space. We found it most convenient to use a UID=0, as from an overall security point of view this is more restrictive than defining the started task as trusted.

```
ALU tcpip_user OMVS(UID(0) HOME(/) PGM(/bin/sh))
```

After these definitions are made, TCP/IP should successfully connect to OpenEdition. In our sample setup, the TCP/IP system address space was started with a user that matched the address space name, T18OTCP. The following message is issued when the TCP/IP system address space connects to OMVS:

```
EZY2140I OpenEdition-TCP/IP connection established for T18OTCP 1
```

The identification **1** which is put into message EZY2140I is the user ID that you assigned to the started task or job under which TCP/IP for MVS runs.

A further definition must be made if *raw* socket support from OpenEdition sockets programs is to be allowed. An example of a program that uses raw sockets is a ping program. In a UNIX environment, raw socket programs are normally restricted to be used only by superusers. This is also true in the OpenEdition environment. In addition, TCP/IP must be informed about who may use raw sockets. Because of the integration of socket I/O into the OpenEdition environment, the socket() call is actually issued from the OMVS address space. Therefore the name of this address space has to be added to the *obey list* in your TCP/IP for MVS PROFILE data set as shown in Figure 31.

```
;   ****************************************************************
;   * ALLOW USE OF OBEYFILE COMMAND FOR FOLLOWING USERS:           *
;   ****************************************************************
;
OBEY
  T18OROUT                  ; ROUTED
  ALFREDC                   ; Alfred
  OMVS                      ; OMVS Raw sockets
ENDOBEY
;
```

Figure 31. Obey List Extension for Raw Socket Support

If you want to allow non-superusers to execute raw socket programs, like ping, you can from a superuser turn on the set-user-id bit of the ping program. By doing so, the ping program will always run with superuser privileges no matter who starts it:

```
/usr/sbin: >chmod u=s,g=s,o=rx /usr/sbin/ping
/usr/sbin: >ls -al ping
---S--Sr-x   1 OMVSKERN DCEGRP     98304 Jun  1 15:54 ping
```

Note: There is no ping program supplied with either OpenEdition or the TCP/IP for MVS OpenEdition Applications Feature. It has proven to be a very simple matter to port the C source code of ping into the OpenEdition environment and make it work as a normal OpenEdition socket program.

2.7 FTPD

The use of an FTPD server brings up a special security aspect. It is quite common to provide FTP services for unidentified users, the so-called ANONYMOUS user that logs in with a user ID of ANONYMOUS. The OpenEdition FTPD server supports both fully identified users and the anonymous user. The anonymous user support is a configuration option. You have to decide if you allow it or not, and if you do allow it, you have to specify how the support is going to be implemented in your environment.

2.7.1 Identified User

The FTPD listener is a never-ending server process that waits for connection requests on the well-known port number 21. After an FTP client connects, the FTPD listener process starts another process that is going to be used for this individual FTP session. In our setup, the FTPD listener process was started with a user ID of OMVSKERN (see Figure 32).

```
RESPONSE=SA18
 BPXO001I 16.15.13 DISPLAY OMVS 879
 OMVS     ACTIVE                                          BPXPRM18
 USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
 OMVSKERN OMVSINIT 0036          1          0 MRI   09.17.28       .201
   SERVER=Init Process                     AF=    0 MF=65535 TYPE=FILE
 OMVSKERN FTPD5 2  0031    2162690    5898252 1FI   16.15.05       .280
   LATCHWAITPID=         0 CMD=/usr/sbin/ftpdsrv 6144 0 15 1 80 0 255
 OMVSKERN SYSLOGD4 0035      65539          1 1FI   09.17.58     37.624
   LATCHWAITPID=         0 CMD=/usr/sbin/syslogd
 T18OTCP  T18OTCP  0044     131076          1 MR    09.18.22    983.086
 OMVSKERN INETD4   0030     262151          1 1FI   14.48.12      1.292
   LATCHWAITPID=         0 CMD=/usr/sbin/inetd
 ALFREDC  ALFREDC  0040      65544          1 1RI   14.47.31     14.670
 HDM      HDM      0041     458761          1 MR    12.59.58    138.790
 OMVSKERN FTPD1 1  0037    5898252          1 1FI   09.21.19       .728
 HDM      HDM      0041    5963790     458761 1C    09.00.04      5.620
   LATCHWAITPID=         0 CMD=sh -L
 TCPW3SRV TCPW3SRV 0042    2031631          1 HFI   14.43.46       .773
```

Figure 32. FTPD Server Before User Enters Login Information

1 This is the FTPD listener process that executes the ftpd program.

2 This is the new process that handles the new client connection. The program in this process is the ftpdsrv program. At this time, the FTP client user has not yet entered a user ID and password, which is the reason why this process still executes under the same user ID as the FTPD listener process.

```
RESPONSE=SA18
 BPXO001I 16.16.04 DISPLAY OMVS 899
 OMVS     ACTIVE                                              BPXPRM18
 USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
 OMVSKERN OMVSINIT 0036          1          0 MRI   09.17.28        .201
   SERVER=Init Process                     AF=    0 MF=65535 TYPE=FILE
 HDM      FTPD5 1  0031    2162690    5898252 1FI   16.15.05        .384
   LATCHWAITPID=         0 CMD=/usr/sbin/ftpdsrv 6144 0 15 1 80 0 255
 OMVSKERN SYSLOGD4 0035      65539          1 1F    09.17.58     37.653
   LATCHWAITPID=         0 CMD=/usr/sbin/syslogd
 T18OTCP  T18OTCP  0044     131076          1 MR    09.18.22    983.344
 OMVSKERN INETD4   0030     262151          1 1FI   14.48.12      1.292
   LATCHWAITPID=         0 CMD=/usr/sbin/inetd
 ALFREDC  ALFREDC  0040      65544          1 1RI   14.47.31     14.671
 HDM      HDM      0041     458761          1 MR    12.59.58    138.840
 OMVSKERN FTPD1 2  0037    5898252          1 1FI   09.21.19        .728
 HDM      HDM      0041    5963790     458761 1C    09.00.04      5.620
   LATCHWAITPID=         0 CMD=sh -L
 TCPW3SRV TCPW3SRV 0042    2031631          1 HFI   14.43.46        .775
```

Figure 33. FTPD Server after User Has Entered Login Information

1 After the user enters a user ID and password, the process changes its user ID.

Note: You may expect to find the same kind of information through a D ASCH,ALL command. This is not true; only the kernel address space OMVS keeps track of the real user.

2 Again you can verify the relationship between parent and child processes by picking up the PPID of the user process and the PID of the server process.

Note: Please note that *all* users of the OpenEdition FTPD server must have a valid OMVS segment in their RACF user profiles. This is true even when the user only wants to access MVS data sets via the OpenEdition FTPD server.

If the FTP client end-user maneuvers in a way he or she is not allowed to, the file system and RACF immediately notice it, the request fails, and a corresponding RACF message is issued on the MVS console (see Figure 34).

```
ICH408I USER(HDM     ) GROUP(OMVSGRP ) NAME(HANS MERTIENS       )
  . CL(FSOBJ   ) FID(01E2D4E2E5F0F2000110000000000003)
  INSUFFICIENT AUTHORITY TO OPENDIR
  ACCESS INTENT(R--)  ACCESS ALLOWED(GROUP --X)
```

Figure 34. FTP User Attempt to Violate Access Authorization

Notes:

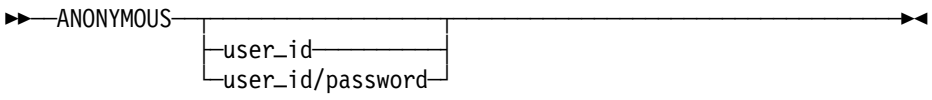
1. CL(FSOBJ) identifies an internal RACF class. It refers to a file system object. You do not need to define it anywhere, it is used for auditing purposes. You may see the following classes in similar messages: DIRACC, DIRSRCH, FSSEC, IPCOBJ, PROCACT, PROCESS.
2. FID identifies the file system object in question.

2.7.2 Anonymous User

A special situation with respect to user identification occurs with FTP. This service may be accessed by a so-called ANONYMOUS user. The OpenEdition FTPD server must be specifically configured to support anonymous access. The default option is to refuse anonymous logon.

The anonymous support can be enabled in three different flavors. The definitions are made in the FTPD server's FTP.DATA configuration data set or file. The syntax for defining anonymous access is as follows:

Defining Anonymous FTP Access



The three flavors are defined as follows:

(no options) When an FTP client enters a user ID of anonymous, the FTPD server uses an MVS user ID of ANONYMOU. The user will be prompted for a valid password. If this user ID has not been defined in RACF or if the user ID has been defined but no OMVS segment created, the logon request fails.

user_id When the FTP client user enters a user ID of anonymous, the FTPD server will act as if the user had entered the specified user ID and the user will be requested to enter the correct password. All access to resources will be made based on the specified user ID. If this user ID has not been defined in RACF or if the user ID has been defined but no OMVS segment created, the logon request fails.

user_id/password When the FTP client user enters a user ID of anonymous, the FTPD server will act as if the user had entered the specified user ID, but the user will not be requested to enter a password. The FTPD server will use the specified password in its request to RACF to create a user security environment.

From the previous description it follows immediately that true support of anonymous users is only possible in the last case. That means you have to add an anonymous definition to your FTP.DATA configuration data set or file (see: Figure 35).

```
ANONYMOUS userid/password
```

Figure 35. Real Anonymous User Support in FTPD

In our sample setup, we defined a user ID called FTPANOM:

```
ADDUSER  FTPANOM  OMVS(UID(999) HOME(/u/ftpanom) PGM(/bin/sh))
```

Figure 36. Defining Anonymous RACF User ID

To limit access to resources in the Hierarchical File System, we connected this user to the same RACF group we used for public access to our MVS Web server, the EXTERNAL group.

User FTPANOM has no TSO segment defined, so even if someone was able to read FTP.DATA where the password has to be stored in clear text, no TSO access is possible. If the password was compromised, another possible entry into MVS could be through rlogin or telnet. To manage that situation, we defined a profile in the /u/ftpanom directory that just contained an exit command (see Figure 37).

```
echo Hello $LOGNAME.
echo Sorry, you do not have access to the system.
exit
```

Figure 37. The .profile File for the Anonymous FTP User

Further we removed ownership of .profile, assigned OMVSKERN as the owner, and modified the attribute to 045, which restricts FTPANOM to just reading and executing this file. To write a shell history log, we gave .sh_history attributes of 060 so FTPANOM can write to it.

```
pwd=/u/ftpanom: >ls -nal
total 24
drwx------   2 999      999            0 May  7 01:18 .
drwxr-xr-x  13 0        0              0 May  7 00:58 ..
----r--r-x   1 0        999           69 May  7 01:08 .profile
----rw----   1 999      999           14 May  7 01:48 .sh_history
-rw-r-----   1 999      999           57 May  7 01:17 test.scr
```

A sample FTP logon sequence looks as follows:

```
[C:\]ftp mvs18o
IBM TCP/IP for OS/2 - FTP Client ver 15:51:28 on Nov 19 1994
Connected to mvs18o.itso.ral.ibm.com.
220-FTPD1 IBM MVS V3R1 at mvs18oe.itso.ral.ibm.com, 17:44:08 on 1996-05-28.
220 Connection will close if idle for more than 5 minutes.
Name (mvs18o): anonymous
230 'ANONYMOUS' logged on.  Working directory is "/u/ftpanom".
ftp> dir
200 Port request OK.
125 List started OK
total 8
-rw-r-----   1 FTPANOM  EXTERNAL      57 May  6 23:17 test.scr
250 List completed successfully.
73 bytes received in 0.09 seconds (0 Kbytes/s)
ftp>
```

2.7.3 FTPD Server Security Exit Routines

The FTPD server allows you to implement several security-related exit routines. See *TCP/IP V3R1 for MVS Implementation Guide*, GG24-3687 for details on these exits.

The OpenEdition version of the FTPD server is stored in the Hierarchical File System and in the TCP/IP for MVS SEZALINK library. As we mentioned earlier,

the OpenEdition FTPD server is implemented in two load modules: the *ftpd* and the *ftpdsrv* modules. As you can see in Figure 38 on page 63, both modules have the sticky bit set.

```
pwd=/usr/sbin: >ls -al ft*
-rwxr-xr-t   2 BPXROOT  DCEGRP    311296 Apr 22 20:22 ftpd
-rwxr-xr-t   2 BPXROOT  DCEGRP    782336 Apr 24 14:18 ftpdsrv
pwd=/usr/sbin: >
```

Figure 38. Directory Entries for the FTPD Server Modules

Consequently both modules are not loaded from the Hierarchical File System. They are loaded from the TCP/IP for MVS SEZALINK library. As we described in 2.1, “Program Control” on page 50, this library must be protected by program control if you are using BPX.DAEMON security.

If you implement the FTPD server security exits, you must make sure that these modules are located in a program controlled library too. The FTPD server modules load the security exits when they are needed. If these exit routines are located in a library that is not program controlled, MVS sets the dirty bit and the FTPD server modules will not be able to change the user security environment as required. If you, inadvertently place the FTPD server security exits in a wrong library, the FTP client end-user will receive an error message, and the following messages will appear in your SyslogD destination for the *daemon* logging facility name:

```
parse_cmd: entering parse_cmd.
get_command: select rc is 1
get_command: received 10 bytes
parse_cmd: calling user exit FTCHKCMD with: rc 0, numparms 3, userid '',
                                            cmd 'USER' args '3/hdm'.
parse_cmd: return from FTCHKCMD with rc: 0.
user: user routine entered with username 'hdm'.
get_command: select rc is 1
get_command: received 15 bytes
parse_cmd: calling user exit FTCHKCMD for PASS cmd.
parse_cmd: return from FTCHKCMD with rc: 0.
pass: pass routine entered.
pass: calling user exit FTCHKPWD with: rc 0, numparms 3, userid 'HDM'.
pass: return from chkpwd with rc: 0.
pass: termid is '091868D0'.
pass: return from racf with: saf: 0, racf: 0,racf reason: 0 acee: 006F10D8.
pass: calling delacee to delete useracee 006F10D8.
pass2: seteuid failed (157/0B7F02AF) : EDC5157I An internal error has occurred.
```

Figure 39. Daemon Error Log for FTPD

Note: We have discarded the time stamp, process ID, etc. from the above lines.

The errno 157 means that an MVS environmental or internal error occurred. The errno_junior 02AF means that the address space has become dirty and the specified function is not supported in a dirty address space.

2.8 Internet Connection Server for MVS

The most complex implementation as seen from a security standpoint is the implementation of the Internet Connection Server for MVS. The server started task user ID must first of all be assigned a UID=0 in order to have the proper server attributes. In our sample setup, the server started task is called TCPW3SRV, and we defined and assigned a started task user ID of TCPW3SRV with a UID=0 in the OMVS segment.

```
ADDUSER TCPW3SRV DFLTGRP(TCPW3GRP) -
        OMVS(UID(0) HOME('/u/tcpw3srv') PROGRAM('/bin/sh'))
```

As the Internet Connection Server for MVS provides service to “world” type users, you have to pay special attention to the security setup of this server in order to protect your system from malicious users. It seems to be a contradiction to run the server with a user ID that has a UID=0 assigned to it and have a safe environment at the same time. This is where another RACF facility class resource, the BPX.SERVER resource, comes in handy.

Until now we have dealt with how to control a server program′s ability to change the user identity based on an end-user that enters a valid MVS user ID and password. We control this ability via the BPX.DAEMON facility class resource.

Now consider the following: the Web server is requested to fetch resources in MVS, for example, Web pages in the Hierarchical File System, from an end-user that is not defined as a user of our MVS system, which means that there is no user profile in RACF for this user. Should we allow the Internet Connection Server for MVS to access resources on behalf of such a user under the default server identity with a UID=0? This is probably not always a good idea, especially not if we allow such a user to run CGI programs that are executed in processes that are started from the Web server process via spawn() services. What we need is a way to assign an MVS user ID to threaded requests that are executed by users that are unknown on our MVS system. The Internet Connection Server for MVS uses the existing RACF surrogate user support for this purpose, and we control the use of this support from a multithreaded server program via the BPX.SERVER resource and surrogate resource profiles in RACF.

You control, via your definitions in the Web server configuration file httpd.conf, how you want the Web server to handle security-related tasks. From a very high-level perspective, the Web server addresses the following two questions for every request it receives:

1. Does the user want to access a protected resource, for which you have defined that the Web server must verify the identity of the user by requesting a user ID and password from the Web user?

   If this is the case, you also define how the user is going to be authenticated. You have more options available, ranging from creating special Web server password files with user IDs and passwords that are unknown to RACF to letting the Web server pass the received user ID and password to RACF for verification. So even if you do decide to request user identification from the end-user, you do not necessarily have to define all Web users in RACF. For many general requests, you most likely do not want to burden the end-user with a request to enter a user ID and a password, but allow the user access to your public information without knowing who he or she is.

2. Under which user security environment do you want the Web user′s request processed?

   Independent of the previous step, you now have to decide how to actually process the request from the Web end-user. If the user identified him- or herself with a valid RACF user ID and password, you have the option of processing the request under that user′s security environment. But you may also, even if the user is a known user on this MVS system, decide to process the request under another user ID, one of the surrogate users that have been defined for use by the Web server. If the user did not identify him- or herself at all, or if you used the Web server password files to authenticate the user, you have no other option than to process the request under a surrogate user ID, because the Web end-user does not have a known RACF identity.

Notes:

1. For the following discussion we assume that you have set up your environment with respect to program control and BPX.DAEMON security as we described earlier in this chapter. In this section we focus on how to use the BPX.SERVER support.
2. The Web server′s use of RACF facilities provides a high degree of security, but you should not forget that both user IDs and passwords flow through the Internet with only very simple encryption. If you want to have a secured flow of passwords and data you need either a Secured Socket Layer (SSL) or a Secured Hypertext Protocol (S-HTTP); both are considered future extensions of the Internet Connection Server for MVS product.

At first you will have to enable the Web server′s use of surrogate users. Assuming the Web server address space is started with a started task user ID of TCPW3SRV, you need to make the following RACF definitions:

```
RDEFINE  FACILITY BPX.SERVER UACC(NONE)
PERMIT   BPX.SERVER CLASS(FACILITY) ID(TCPW3SRV) ACCESS(UPDATE)
SETROPTS RACLIST(FACILITY) REFRESH
```

To create the surrogate RACF definitions, perform the following steps:

1. Activate the RACF SURROGAT class if it has not been done before.

   ```
   SETROPTS CLASSACT(SURROGAT)
   ```

2. Add the surrogate user(s) and group(s) you need. As a minimum, you need to define a public user ID, which can be used by the Web server to process requests from users that are not defined on this MVS system. In this example, we define a surrogate user ID called TCPW3PUB in a RACF group called EXTERNAL.

   ```
   ADDGROUP EXTERNAL SUPGROUP(superior_group) OMVS(GID(999))
   ADDUSER TCPW3PUB DFLTGRP(EXTERNAL) -
        OMVS(UID(998) HOME('/')  PROG('/bin/sh')
   ```

   Do this for every surrogate user ID that you want to use from your Web server.

3. Next, you have to allow the Web server to use these user ID(s) as surrogate user ID(s).

```
RDEFINE SURROGAT BPX.SRV.TCPW3PUB UACC(NONE) 1
PERMIT BPX.SRV.TCPW3PUB CLASS(SURROGAT) ID(TCPW3SRV) ACCESS(READ) 2
```

1 This command defines the TCPW3PUB as a surrogate user ID.

2 This command allows the Web server that executes with the started task user ID of TCPW3SRV to use TCPW3PUB as a surrogate user ID.

The above definitions will allow you to set up a simple Web server configuration with just a single surrogate user definition. For internal Web servers, this will most likely be sufficient. If you need to group your public access based on different access privileges, you will have to define more surrogate users with varying access privileges to your Web server resources and set up definitions in your Web server configuration file to assign different surrogate user IDs to requests based on the resources that are being accessed.

Please consider carefully the privileges you assign to the surrogate user IDs. Web server requests for resources are going to be authorized via the privileges that are assigned to the surrogate user ID, and you probably do not want public Web users to be able to read, for example, your /etc directory or other important system information in your OpenEdition environment.

2.9 Accounting

Accounting does not have anything to do with security. But, as you are going to define a lot of new user IDs and, more important, old user IDs may be used in a different environment where standard accounting information does not apply, we feel it is important to head you to the place where you define accounting information. This place is the RACF user segment.

Two major OpenEdition services rely on APPC/MVS: fork() and spawn(). As we explained in 1.3.1, “OpenEdition Process” on page 9, both create new processes which in MVS are backed by MVS address spaces. APPC/MVS is designed to handle APPC transaction program requests by MVS directly. APPC/MVS has its own scheduler, which runs in the ASCH address space. It is the ASCH that finally selects an initiator (ASCHINT) address space to schedule the transaction program request assuming the definitions define the ASCH as the scheduler. Somehow you can compare the ASCH with JES/2 or JES/3.

The OE Kernel address space that finally receives a fork() or spawn() service request transforms this request into an APPC ATTACH request.[1] The forked or spawned process is then started in an ASCHINT address space.

To achieve correct accounting for forked or spawned processes, we have to provide the ASCHINT address space with correct accounting information. This is done by means of the WORKATTR segment of the user profile in RACF.

[1] Due to performance reasons, the OE Kernel address space actually talks directly to the ASCH, thereby surpassing the communication with the APPC address space.

```
ADDUSER DILBERT DFLTGRP(ENGNGP7) NAME('THE DILBERT') PASSWORD(A4B3C2D1) -
        OMVS(UID(314) HOME('/u/dilbert') PROGRAM('/bin/sh'))           -
        TSO(ACCTNUM(12345678) DEST(P382005) PROC(PROC01) SYSOUTCLASS(A))-
        WORKATTR(WANAME('THE DILBERT') -
        WAACCNT(12345678) -
        WABLDG(ManyNewspapers) WAROOM(PAGE) -
        WADEPT(COMICS_Departement) -
        WAADDR1(WIDGET_INC) WAADDR2(NEW_YORK) -
        WAADDR3(NEW_YORK) WAADDR4(10002))
```

Figure 40. Work Attribute Segment in User's RACF Profile

Forked address spaces are used in different environments. Some forks require the accounting data to be propagated from the parent, while other forks require the accounting data to be obtained from the WAACCNT field in the WORKATTR segment of the RACF user profile. Because of these two conflicting requirements, the fork code dynamically determines how to tailor the accounting data; therefore, the value specified for TAILOR_ACCOUNT is ignored. Fork performs similar tailoring for SYSOUT information.

You need to specify the account and SYSOUT information in the WORKATTR segment of the RACF user profile if you plan on having users connect to the system using methods not traditionally used on MVS (for example, rlogin, telnet, REXEC, or RSH).

2.10 Summary

The following table gives a summary of all definitions you have to make to run OpenEdition servers with MVS-like security.

Table 6. Overview of Processes, UIDs and RACF Facility Classes

Process/Job	UID	Access on BPX.SERVER	Access on BPX.DAEMON
OMVS	0	NONE	READ
TCP/IP system address space	0	NONE	NONE
InetD	0	NONE	READ
SyslogD	0	NONE	NONE
FTPD	0	NONE	READ
Client user	Not 0	NONE	NONE
Web Server	0	UPDATE	READ
Web User	Not 0, not UID of administrator, not in same group	NONE	NONE

Note: Please remember that as soon as you have defined BPX.DAEMON, program control *must* have been set up correctly and *must* be active. If not, all server requests are going to fail with various error messages. Whenever

something does not work as expected, make it a rule to look at your MVS syslog to see if there are any security violation messages that might indicate a security definition problem.

Chapter 3. Setting Up the OE Resolver

Although it is possible to share resolver configuration data between the OE resolver and the TCP/IP for MVS or AnyNet MVS resolvers, we recommend that you set up a separate set of resolver configuration data sets or files for each of these resolvers. The OE resolver is part of Language Environment and is a separate entity, different from the TCP/IP for MVS resolver and the AnyNet MVS resolver. The OE resolver is used by OpenEdition socket applications, the TCP/IP for MVS resolver is used by native TCP/IP for MVS socket applications, and the AnyNet MVS resolver is used by native AnyNet MVS socket applications. By *native* we mean socket applications that run directly on top of the TCP/IP for MVS or AnyNet MVS protocol stack without using any OpenEdition functions at all.

This chapter includes the following topics:

Please refer to E.2, “OpenEdition Configuration and Files” on page 234 for sample OE resolver configuration files.

© Copyright IBM Corp. 1996

3.1 Resolver Data Formats

The three resolvers support more configuration data formats for some of the data sets or files. See Table 7 for an overview of the supported formats.

Table 7. Supported Resolver Configuration Data Formats

Configuration data		OE	TCP/IP	AnyNet
Name	**Format**			
TCPIP.DATA or RESOLV.CONF	Text (BSD format)	√	√	√
PROTOCOL	Text (BSD format)	√	√	√
SERVICES	Text (BSD format)	√	√	√
HOSTS	Text (BSD format)	√		√
	Binary (output from MAKESITE)	√	√	√
TRANSLATE table	Text			√
	Binary (output from CONVXLAT)	√	√	√

Only the OE resolver offers you the option of placing the configuration files in your Hierarchical File System. Neither the TCP/IP for MVS resolver or the AnyNet MVS resolver is able to read configuration data from the Hierarchical File System. If you do plan to share resolver configuration data among the different resolvers, you must place all configuration data in traditional MVS data sets.

Both TCP/IP for MVS and AnyNet MVS supply a set of sample resolver configuration data sets that you can use as a base for creating your OE resolver configuration data sets or files.

Table 8. Sample Resolver Configuration Data

File	TCP/IP for MVS	AnyNet MVS
TCPIP.DATA or RESOLV.CONF	*datasetprefix*.SEZAINST(TCPDATA)	SYS1.SAMPLIB(RESOLV)
PROTOCOL	*datasetprefix*.SEZAINST(PROTO)	SYS1.SAMPLIB(PROTOCOL)
SERVICES	*datasetprefix*.SEZAINST(SERVICES)	SYS1.SAMPLIB(SERVICES)
HOSTS	*datasetprefix*.SEZAINST(HOSTS)•	SYS1.SAMPLIB(HOSTS)
Translate table	*datasetprefix*.SEZATCPX(OEMVS311)	N/A

Note:

1. Please note that the text format of the TCP/IP for MVS sample for the local host file does *not* match the format that is required by the OE resolver. If you base your OE resolver local host file on this sample, you will have to edit it before the OE resolver is able to use it.

The translate table sample from TCP/IP for MVS must be processed by a TCP/IP for MVS utility called CONVXLAT to produce a binary form of the translate table. The OE resolver supports this binary form, but not the text form of the translate table. If you do not have TCP/IP for MVS available, you can not create the required binary form of a translate table, but both the OE resolver and the AnyNet MVS resolver will work without it; they both have a default built-in translate table available in case no translate table data set has been configured.

All the other configuration data sets or files for the OE resolver can be created based on samples from either TCP/IP for MVS or AnyNet MVS.

In the following sample setup, we chose to place as many of the OE resolver configuration files in the Hierarchical File System as supported.

3.2 /etc/resolv.conf

The resolver configuration data set or file is where you specify, among other things, if the OE resolver should use a name server or not.

By default, the OE resolver will look for /etc/resolv.conf and use that as the resolver configuration file. The contents of a sample /etc/resolv.conf file is shown in Figure 41.

The DATASETPREFIX keyword is used if the OE resolver has to allocate any standard MVS data sets. In the sample setup that is documented in this book, the OE resolver needs this prefix in order to allocate the standard TCP/IP for MVS translation table data set (*datasetprefix*.STANDARD.TCPXLBIN).

You may use the exact same syntax in your /etc/resolv.conf file as you use in your TCP/IP for MVS TCPIP.DATA data set. You should not include TCPIPJOBNAME or TCPIPUSERID in your /etc/resolv.conf file. These parameters have no meaning in the OpenEdition environment.

```
;*********************************************************************
;                                                                    *
;     /etc/resolv.conf                                               *
;                                                                    *
;     OpenEdition MVS Resolver on mvs18                              *
;                                                                    *
;     This file is used by the OpenEdition MVS resolver code in      *
;     all OE socket applications.                                    *
;                                                                    *
;     There is no TCPIPJOBNAME in this file.  The AF_INET PFS        *
;     decides on it's own if it should use a TCP/IP stack or an      *
;     AnyNet stack, and in the case of more TCP/IP stacks, which one.*
;                                                                    *
;*********************************************************************
;
Datasetprefix TCPIP.OMVS       ; Only used for TCPXLBIN data set
Messagecase mixed              ;
HostName mvs18oe               ; MVS18 OE environment host name
DomainOrigin itso.ral.ibm.com  ;
NSinterAddr  9.24.104.126      ; Use our own caching-only DNS first
NSinterAddr  9.24.104.108      ; Then go to rsserver.itso.ral.ibm.com
NSportAddr 53                  ;
ResolveVia UDP                 ;
ResolverTimeout 10             ; Don't wait too long
ResolverUdpRetries 1           ;
```

Figure 41. Sample /etc/resolv.conf File

Please note that the OE resolver will use the DOMAINORIGIN value from the OE resolver configuration file, but it will use the HOSTNAME of the resolver configuration data set from your default AF_INET transport provider. If you have

defined your TCP/IP for MVS stack as the default AF_INET transport provider and have defined a TCPIP.DATA data set for this stack with the HOSTNAME and DOMAINORIGIN statements shown in Figure 42 on page 72, an OE socket application will receive host name HOOLIGAN on a gethostname() call, and a gethostbyname() call for host name friendly will be sent to the name server at 9.24.104.126 querying the fully qualified name of friendly.itso.ral.ibm.com based on the DOMAINORIGIN and name server IP address from the /etc/resolv.conf file.

```
 ...
HostName hooligan
DomainOrigin your.neck.of.the.woods.com
NSinterAddr  192.168.1.1
 ...
```

Figure 42. Sample Extract of TCPIP.DATA of Default Stack

Apart from the host name that is returned on a gethostname() call, all information used by the OE resolver comes from the OpenEdition resolver configuration file.

If an application wants to override the system default of /etc/resolv.conf, it can initialize the RESOLVER_CONFIG environment variable to point to an alternate resolver configuration file or data set. If you are logged on to the OpenEdition shell, the following command can be used to point to an MVS data set:

```
export RESOLVER_CONFIG="//'TCPIP.ITSC.CNTL(TCPDATA)'"
```

Please note the required syntax for referring to an MVS data set name: double quotes followed by two slashes followed by the MVS data set name enclosed in single quotes. You will have to use the same syntax for other environment variables that point to MVS data sets.

The following shell command can be used to point to an HFS file:

```
export RESOLVER_CONFIG=/etc/testresolv.conf
```

You might want to use these options if you have a requirement for different servers to use, for example, different name servers. If you implement, for example, both a Web server for internal use and a Web server for external use, you can point the Web server for external use to a different name server than the one you use internally.

3.3 /etc/protocol

The /etc/protocol file holds information about supported protocols and is used by the following resolver calls:

Getprotobyname	Get protocol number based on a protocol name
Getprotobynumber	Get protocol name based on a protocol number
Setprotoent	Prepare to read the protocol file sequentially
Getprotoent	Read next protocol file entry
Endprotoent	End reading the protocol file sequentially

The protocol number is used in the 8-bit protocol field in an IP packet header.

To create the OE resolver /etc/protocol file, we used the sample protocol member from *datasetprefix*.SEZAINST(PROTO), which has the following content (some comments removed for readability):

```
# offical name, protocol number, aliases

ip              0               # dummy for IP
icmp            1               # control message protocol
tcp             6               # tcp
udp             17              # user datagram protocol
```

Figure 43. /etc/protocol Sample

3.4 /etc/services

The /etc/services file holds information about how individual applications are assigned to socket layer port numbers. Standard applications, like telnet or FTP, are assigned port numbers inside the well-known port number range (from 0 to 1023). You can assign port numbers to your own server applications by adding entries to the /etc/services file. The content of /etc/services is typically used by a server program via a getservbyname() call, where the server program passes its own name and receives the assigned port number when the call returns. The server program can then bind its socket to the assigned port number. This technique allows you to keep port number assignments external to your server program. This is of particular importance if you want to start more instances of your server program on the same TCP/IP stack. By using different /etc/services files, each instance of the server program may be assigned alternate port numbers.

The /etc/services file is used by the following resolver calls:

Getservbyname Get server port number based on server name

Getservbynumber Get server name based on server port number

Setservent Prepare to read /etc/services sequentially

Getservent Read next entry in /etc/services

Endservent End reading /etc/services sequentially

The port number is used on various socket calls and is also included in both the header of a TCP segment and the header of a UDP datagram.

Figure 44 on page 74 is an extract of the /etc/services file.

```
# Name          Port/protocol   Aliases

echo            7/tcp
echo            7/udp
discard         9/tcp           sink null
discard         9/udp           sink null
systat          11/tcp          users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
qotd            17/tcp          quote
chargen         19/tcp          ttytst source
chargen         19/udp          ttytst source
ftp             21/tcp
telnet          23/tcp
smtp            25/tcp          mail
time            37/tcp          timserver
time            37/udp          timserver
rlp             39/udp          resource        # resource location
nameserver      42/tcp          name            # IEN 116
whois           43/tcp          nicname
domain          53/tcp          nameserver      # name-domain server
domain          53/udp          nameserver

(more entries in actual /etc/services)
```

Figure 44. /etc/services Sample (Extract)

In general, servers use a getservbyname() call to find the assigned port number. Some servers allow you to override the port number via a server-specific configuration or start option. The OE FTPD server, for example, allows you to pass a run-time option:

```
PORT 7021
```

If this option is specified, the FTPD server will not use the value assigned in /etc/services, but use the value specified in the run-time option. You can use this technique to start alternate FTPD server instances on alternate port numbers.

Servers that are started via InetD use the service name, which you specify in the /etc/inetd.conf file as argument on the getservbyname() call. If you, for example, want to have your OE TelnetD server to operate on port 2023 instead of the default port 23, you can edit the line in your /etc/services file that corresponds to the telnet service name in your /etc/inetd.conf file.

```
/etc/inetd.conf (extract):
--------------------------

telnet   stream tcp nowait OMVSKERN /usr/sbin/otelnetd otelnetd

/etc/services (extract):
------------------------

telnet        2023/tcp
```

Figure 45. Assigning Port Number to the TelnetD Server

If you want to start two telnet servers in your OE environment, you can use the following set of definitions:

```
/etc/inetd.conf (extract):
--------------------------

telnet     stream tcp nowait OMVSKERN /usr/sbin/otelnetd otelnetd
testtelnet stream tcp nowait OMVSKERN /usr/test/otelnetd otelnetd

/etc/services (extract):
------------------------

telnet          23/tcp
testtelnet    2023/tcp
```

Figure 46. Assigning Port Numbers to Two TelnetD Servers

With this setup, you will have your normal OE telnet server operating on the standard port 23, and you will have your test telnet server operating on port 2023. If a telnet client connects to port 2023, the telnet client will use your test telnet server.

3.5 /etc/hosts

You can set up the local hosts file to support local host name resolution. If you use the local hosts file for this purpose, your socket applications will only be able to resolve names and IP addresses that appear in your local hosts file.

If you need to resolve host names outside your local area, you can configure the OE resolver to use a domain name server (see 3.2, “/etc/resolv.conf” on page 71). If you use a domain name server, you do not need to set up any host definitions in your OE resolver configuration, but you may, however, still do so.

If you have configured your OE resolver to use a name server, it will always try to do so, unless your applications were written with a RESOLVE_VIA_LOOKUP symbol in the source code. If this is the case, all name resolution calls from such a program will always use the local hosts file. This is probably not a technique you will see for standard socket applications, but it may be a technique you could find use for when you develop your own socket applications for the OpenEdition environment.

It may also be a good idea to have some basic local hosts file available for the OE resolver to use if the name server is not reachable. If the name server does not respond to name resolution requests, the OE resolver will try to use the local hosts file.

If the name server is reachable but returns a negative reply for a name resolution request, the OE resolver will try to resolve the unqualified name via the local hosts file, if such a file is present. Assume you try to resolve the host name *friendly* and your DOMAINORIGIN is *my.wood.com*, the OE resolver will send a query to the name server for *friendly.my.wood.com*. If the name server returns a negative reply (the name is not registered), the OE resolver will look into the local hosts file for an entry of friendly and, if not found, for an entry of friendly.my.wood.com.

Due to the higher flexibility of the Domain Name System, we recommend you use a domain name server. If you have TCP/IP for MVS installed, you can configure and use the name server that comes with that product. You need to configure your OE resolver configuration file to point to the IP host, where your name server is running. It may be any host that is reachable from your OpenEdition environment. There is no requirement that the name server must run on MVS/ESA. It may, from a functional point of view, run equally well on, for example, an AIX/6000 or an OS/2 TCP/IP host.

If you set up a small TCP/IP network, the simplicity of the local hosts file approach is preferable.

The following resolver calls are serviced either via the local hosts file, or via requests sent by the OE resolver to a name server:

Gethostbyname Resolve a host name into one or more IP addresses

Gethostbyaddr Resolve an IP address into a host name

The following resolver calls only apply to your local hosts file. If you have configured your system to use a name server, these calls will bypass the name server and use your local hosts file, if you have configured one. If no local hosts file exists, the sethostent() call will return with an error.

Sethostent Prepare to read your local hosts file sequentially

Gethostent Read next entry in your local hosts file

Endhostent End reading your local hosts file sequentially

The OE resolver supports two different formats for the local hosts file:

1. The standard BSD formatted text file - as it is supported in most TCP/IP implementations.
2. The format that the TCP/IP for MVS MAKESITE utility program creates. This option is only available if you have TCP/IP for MVS installed. The MAKESITE utility program comes with TCP/IP for MVS.

You can choose freely which format you want to use in your OE resolver environment, with one exception: if your applications rely on one of the following resolver calls, you must use the format that is created via the MAKESITE utility:

Getnetbyaddr Get a net entry by name

Getnetbyname Get a net entry by network address

Setnetent	Prepare to read the net entries sequentially
Getnetent	Get next net entry
Endnetent	End reading net entries sequentially

In most UNIX systems these calls are serviced via a file called /etc/networks, but this file is currently not supported by the OE resolver. If you use the TCP/IP for MVS MAKESITE utility, this utility supports a HOSTS.LOCAL source file according to the RFC952 syntax, which allows you to specify both host and network entries. The resulting files from MAKESITE may therefore hold both host and network entries, which is the reason why these calls are supported if you use the MAKESITE format.

If you want to use the text format, we recommend that you place your local hosts file in the Hierarchical File System under the name /etc/hosts. Please observe that some of the available documentation specifies this name differently (without the last ″s″), which will not work.

See Figure 47 for a sample /etc/hosts file.

```
#
# OE Resolver /etc/hosts file on mvs18oe.
#
# The format of this file is:
#
# Internet Address      Hostname   Aliases     # Comments
#
# Items are separated by any number of blanks and/or tabs.  A '#'
# indicates the beginning of a comment; characters up to the end of the
# line are not interpreted by routines which search this file.  Blank
# lines are allowed in this file.

9.24.104.126    mvs18oe mvsoe # OE host
192.168.210.1   mvs18an       # AnyNet MVS host
192.168.210.8   mypcaa        # AnyNet gw host
9.24.104.79     mypc          # A workstation
```

Figure 47. Sample /etc/hosts File

There are some syntax requirements for the /etc/hosts file. The most important are the following:

1. A host name can have a maximum of four qualifiers:
   - Host name a.b.c.d is a valid host name.
   - Host name a.b.c.d.e is *not* a valid host name.
2. You can specify a maximum of 35 aliases per IP address.
3. A qualifier can have a maximum length of eight characters:
   - Host name mypchost.mynet is a valid host name.
   - Host name myotherpchost.mynet is *not* a valid host name.

   This restriction will be removed when APAR PN79720 is solved. The solution of this APAR will increase the length of each qualifier to 24 characters.

As there is no syntax checking of the /etc/hosts file before an application program tries to use the information in it, syntax errors may show up as error messages from application programs:

```
Error in line 2: A name qualifier must be 8 characters or less
sethostent() returned error:
             EDC5049I The specified file name could not be located.
```

The above error message was returned due to the following contents of an /etc/hosts file:

```
9.24.104.126    mvs18o  mvsoe  # OE host
192.168.210.1   mvs18antoolong # AnyNet MVS host
192.168.210.8   mypcaa         # AnyNet gw host
9.24.104.79     mypc           # A workstation
```

If you already have a TCP/IP for MVS stack implemented and want to maintain your local hosts file in one place, you can instruct the OE resolver to use the same file format as TCP/IP for MVS uses.

You can use two different approaches:

1. Maintain your local hosts file in the source format that is accepted by the TCP/IP for MVS MAKESITE utility program. This format is documented in RFC952.
2. Maintain your local hosts file in the BSD source format that is accepted by the OE resolver and most other TCP/IP platforms.

3.5.1 Maintaining Shared Source in HOSTS.LOCAL

You maintain your HOSTS.LOCAL file in the format that is required by the TCP/IP for MVS MAKESITE utility program. See Figure 48 for a sample HOSTS.LOCAL data set.

```
; HOSTS.LOCAL (Input to TCP/IP for MVS MAKESITE utility)
;
; Syntax requirements documented in RFC952.
;
HOST : 9.24.104.126  : mvs18a, mvsoe ::::
HOST : 192.168.210.1 : mvs18aa ::::
HOST : 192.168.210.8 : mypcaa ::::
HOST : 9.24.104.79   : mypc ::::
HOST : 9.24.104.80   : abc.ibm.com ::::
HOST : 9.24.104.81   : abc1, abc1.ibm.com ::::
;
NET :  9.24.104.0    : itso.ral.ibm.com :
NET :  9.0.0.0       : ibm.com :
```

Figure 48. Sample HOSTS.LOCAL Source

When you run the MAKESITE utility program, it produces two output data sets:

- *datasetprefix*.HOSTS.SITEINFO
- *datasetprefix*.HOSTS.ADDRINFO

You can instruct the OE resolver to use these data sets in two ways:

1. To use these data sets as a system default, you must ensure that there is no file called /etc/hosts in your Hierarchical File System, and in addition, you must in the resolver configuration data set or file (see 3.2, "/etc/resolv.conf" on page 71) specify the DATASETPREFIX keyword with the value of the high-level qualifier of your HOSTS.SITEINFO and HOSTS.ADDRINFO data sets.

   If your DATASETPREFIX in the OE resolver configuration data set or file is TCPIP.OMVS, the OE resolver will use:

   ```
   TCPIP.OMVS.HOSTS.SITEINFO
   TCPIP.OMVS.HOSTS.ADDRINFO
   ```

2. If you want to override your system default for a specific application, you can set the two environment variables called X_SITE and X_ADDR to point to two data sets that are created with the MAKESITE utility. If your application executes in the shell environment, you can use the following commands to assign values to the environment variables:

   ```
   export X_SITE="//'MYOWN.HOSTS.SITEINFO'"
   export X_ADDR="//'MYOWN.HOSTS.ADDRINFO'"
   ```

If you use the same DATASETPREFIX for your TCP/IP for MVS resolver, the same set of HOSTS.SITEINFO and HOSTS.ADDRINFO data sets can be used by both TCP/IP for MVS and OpenEdition.

3.5.2 Maintaining Shared Source in /etc/hosts

TCP/IP for MVS can not directly read the /etc/hosts file, so you have to create HOSTS.SITEINFO and HOSTS.ADDRINFO data sets from your /etc/hosts source file. As the format of the /etc/hosts file is not compatible with the format that is required by the TCP/IP for MVS MAKESITE utility, you need to process the /etc/hosts file with a small home-written REXX program before running the MAKESITE command. The sample REXX program in Appendix A, "Sample REXX to Create HOSTS.LOCAL from /etc/hosts" on page 203 can be used for such a purpose.

3.6 OE Resolver Translation Table

The OE resolver has to translate EBCDIC host names and IP addresses into ASCII before sending a request to a name server, and it has to translate the ASCII response from the name server into EBCDIC before presenting the result to the calling application program.

The OE resolver has a built-in translation table, so you do not have to configure a translation table if the built-in table meets your requirement.

In the sample setup, we created an OE resolver translation table data set from the TCP/IP for MVS supplied sample table that is located in *datasetprefix*.SEZATCPX(OEMVS311). This table assumes an ASCII code page of ISO 8859-1 and an EBCDIC code page of IBM 1047.

The translate table cannot be used by the OE resolver in the supplied source format; you have to use the TCP/IP for MVS utility called CONVXLAT to convert it into a binary form that is acceptable to both TCP/IP for MVS and OpenEdition:

```
convxlat 'tcpip.sezatcpx(oemvs311)' 'tcpip.omvs.standard.tcpxlbin'
```

If your DATASETPREFIX in the OE resolver configuration data set or file has been set to TCPIP.OMVS, the OE resolver will by default use the TCPIP.OMVS.STANDARD.TCPXLBIN translate table data set.

If some applications have specific requirements that are not met by your default translate table data set, you can create alternate data sets and point to an alternate data set by setting the X_XLATE environment variable before using the resolver calls.

3.7 OE Resolver Configuration Summary

The following setup was used in the ITSO Raleigh sample environment:

1. OE resolver:
   - /etc/resolv.conf for the basic OE resolver configuration data. This file includes a DATASETPREFIX keyword with a value of TCPIP.OMVS.
   - /etc/protocol for the protocol configuration data.
   - /etc/services for the OpenEdition socket server port number assignment.
   - /etc/hosts for local host definitions. Primary source for name resolution was a name server. /etc/hosts was only used for backup purposes and to support the resolver calls that bypass a name server.
   - TCPIP.OMVS.STANDARD.TCPXLBIN as the OE resolver ASCII-EBCDIC translate table data set.
2. TCP/IP for MVS resolver:
   - TCPIP.T18OE.TCPPARMS(TCPDATA) for the TCPIP.DATA configuration data. This file includes a DATASETPREFIX keyword with the value of TCPIP.T18OE.
   - TCPIP.T18OE.PROTO for the protocol configuration data.
   - TCPIP.T18OE.SERVICES for the TCP/IP for MVS socket server port number assignment.
   - TCPIP.T18OE.HOSTS.ADDRINFO and TCPIP.T18OE.HOSTS.SITEINFO. These data sets were created from /etc/hosts via the REXX program described in Appendix A, “Sample REXX to Create HOSTS.LOCAL from /etc/hosts” on page 203.
   - TCPIP.T18OE.STANDARD.TCPXLBIN as the TCP/IP for MVS resolver ASCII-EBCDIC translate table data set.
3. AnyNet MVS resolver:
   - TCPIP.ANYNET.RESOLVER for the AnyNet MVS resolver configuration data, which is used to point to a name server for native AnyNet MVS socket applications. Host name and pointers to other AnyNet MVS resolver configuration data sets are specified in the AnyNet MVS environment data set (in our setup: SYS1.VTAMLIST(ENVVAR)).
   - TCPIP.ANYNET.ETC.PROTO for the protocol configuration data.
   - TCPIP.ANYNET.ETC.SERVICES for the AnyNet MVS socket server port number assignment.
   - TCPIP.ANYNET.HOSTS.ADDRINFO and TCPIP.ANYNET.HOSTS.SITEINFO. These data sets were created from /etc/hosts via the REXX program

described in Appendix A, “Sample REXX to Create HOSTS.LOCAL from /etc/hosts” on page 203.

- TCPIP.ANYNET.STANDARD.TCPXLBIN as the AnyNet MVS resolver ASCII-EBCDIC translate table data set.

Please see Appendix E, “Configuration File Samples” on page 227 for the contents of these configuration files and data sets.

Chapter 4. Setting Up the AF_INET Transport Providers

This chapter includes detailed instructions on the options you have available for establishing an AF_INET transport provider configuration with either TCP/IP for MVS or AnyNet MVS or both.

This chapter includes the following topics:

- 4.1, AF_INET Sockets
- 4.2, AF_INET Transport Providers
- 4.3, TCP/IP for MVS, Shared Stack
- 4.4, TCP/IP for MVS, Separate Stacks
- 4.5, AnyNet MVS, Single Stack
- 4.6, TCP/IP for MVS and AnyNet MVS As Transport Providers
- 4.7, AnyNet/2 Setup
- 4.8, Start of OpenEdition, TCP/IP, AnyNet
- 4.9, Stopping OMVS, TCP/IP and AnyNet
- 4.10, Recycling Transport Providers
- 4.11, Automation of OMVS Operations

© Copyright IBM Corp. 1996

4.1 AF_INET Sockets

There is no TCP/IP protocol stack imbedded in OpenEdition. OpenEdition relies on either TCP/IP for MVS or AnyNet MVS to provide a TCP/IP protocol stack that allows OpenEdition socket programs to communicate with socket programs on other hosts in your network using AF_INET sockets.

A socket is the endpoint of a communication path; it identifies the address of a specific process at a specific computer using a specific transport protocol. The exact syntax of a socket address depends on the protocol being used, on its *addressing family*. When you obtain a socket via the socket() system call, you pass a parameter that tells the socket library to which addressing family the socket should belong. All socket addresses within one addressing family use the same syntax to identify sockets; in other words, they belong to the same addressing family.

4.1.1 Socket Addressing Families in OpenEdition

In an OpenEdition environment, you are able to use the following addressing families:

Family **Description**

AF_INET The Internet addressing family. Also referred to as the Internet domain.

This addressing family is used within the TCP/IP domain to identify sockets on IP hosts. A socket address in AF_INET consists of the following:

Family Half-word binary with a value of 2, which identifies the socket address as belonging to the AF_INET addressing family.

Port Half-word binary with port number that identifies the process.

IP address Full-word binary with IP address of IP host in network byte order format.

Reserved 8 reserved bytes.

The following is an example of an AF_INET address that represents the telnet server (port number 23) on an IP host with the IP address of 9.24.104.126:

```
{AF_INET 23 9.24.104.126}
```

AF_UNIX The UNIX addressing family. Also referred to as the UNIX domain.

You can use AF_UNIX with OpenEdition sockets, where this addressing family is used for interprocess communication between OpenEdition processes within one MVS operating system. The syntax of an AF_UNIX address is as the following:

Family Half-word binary with a value of 1, which identifies the socket address as belonging to the AF_UNIX addressing family.

Path 108 characters defining a path name (similar to a Hierarchical File System path name) by which this local process wants to be known by other local processes.

The following is an example of an address in the AF_UNIX addressing family:

```
AF_UNIX /u/xyz/testsrv
```

OpenEdition implements support for a given addressing family through different physical file systems. There is one physical file system for the AF_UNIX addressing family and there is another for the AF_INET addressing family.

The AF_UNIX physical file system is self-contained within OpenEdition and does not rely on other products to implement the required functions.

The AF_INET physical file system does rely on other products to provide the AF_INET transport services. The IBM products that you can use as AF_INET transport providers in the OpenEdition environment are:

- TCP/IP for MVS
- AnyNet MVS

For more information on sockets and socket programming in an MVS environment, please see *A Beginner's Guide to MVS TCP/IP Socket Programming*, GG24-2561.

4.2 AF_INET Transport Providers

When you design your OpenEdition environment, you have a couple of options available for selecting AF_INET transport providers. The decisions you make influence the way you have to customize the OpenEdition environment and the transport providers.

4.2.1 Single or Multiple AF_INET Transport Providers

Do you want to use a single AF_INET transport provider or do you want to use multiple AF_INET transport providers?

If you have your background in a UNIX environment, this may seem to be a strange question to ask, as you are used to the TCP/IP protocol stack being an integral part of the UNIX operating system. This is not the case in an MVS/ESA or OS/390 environment. In this environment you may start more instances of a TCP/IP protocol stack, each stack running on the same operating system, but each stack having a unique TCP/IP identity in terms of network interfaces, IP addresses, host name and socket applications. If you have the requirement to start more TCP/IP stacks, you also need to decide if all your stacks are to be used as OpenEdition AF_INET transport providers or if only one of your stacks is to be used for OpenEdition communication.

A simple example of a situation where you have more TCP/IP stacks running in your MVS/ESA system is when you need to support socket communication over both an IP network through TCP/IP for MVS and an SNA network through AnyNet MVS.

Another example is if you have two separate IP networks, one friendly and the other not so friendly, and you do not want to allow IP routing between them, but you want to give hosts on both IP networks access to your OpenEdition environment. In this situation you could implement two TCP/IP for MVS stacks, one connected to the friendly IP network and another connected to the not so

friendly network. Because the two stacks are separate, IP routing cannot be done between them. An OpenEdition application may act as an application gateway between the two networks, but in that situation all the information exchange between the two networks is under control of that application and the application can limit information exchange to the required level.

If you can do with a single AF_INET transport provider for OpenEdition use, you can select to use the integrated sockets physical file system of OpenEdition. If you need more AF_INET transport providers, you have to use the converged sockets physical file system.

You can customize OpenEdition to use the converged sockets physical file system with just a single AF_INET transport provider, but it is generally not recommended due to a slight performance decrease as compared to the integrated sockets physical file system.

4.2.2 Shared or Separate TCP/IP for MVS Stack

If you decide to use TCP/IP for MVS as an AF_INET transport provider for your OpenEdition socket programs, you also have to consider if you will use a shared TCP/IP for MVS stack implementation or if you will use a separate-stack TCP/IP for MVS implementation, where one stack supports the OpenEdition socket applications and another stack supports the non-OE socket applications.

This is an important point, which you have to consider carefully. The point is best illustrated by an example. In TCP/IP for MVS you now have two telnet servers that each have unique functions:

- The standard TCP/IP for MVS telnet server supports tn3270 and line-mode telnet clients. The line-mode clients can log on to TSO as line-mode terminals.
- The new OE telnet server supports line-mode and raw-mode telnet clients. Both raw-mode and line-mode clients log on to the OpenEdition shell environment.

For convenience, both servers should reside on the well-known telnet server port 23, but that is not possible if both servers run on the same TCP/IP for MVS stack. Only one of them can reside on port 23 and the other one must be assigned to an alternate port number. If you decide to use a single stack and assign, for example, the OE telnet server to port 2023, then all telnet clients that want to connect to your OE telnet server must be told that the OE telnet server port is 2023, and they must include that port number on their telnet client command invocation:

```
telnet -p 2023 mvshostname
```

The same considerations apply to the remote execution and remote shell servers. The non-OE versions of these servers support the ability to execute TSO commands, but not OpenEdition shell commands. The OE versions of these two servers support the ability to execute OpenEdition shell commands, but not TSO commands. If your clients need the ability to do both, you need to start both sets of servers in your MVS/ESA system and you have to decide which of the following two configurations you want to implement:

1. A *separate-stack* implementation, where one stack is used with non-OE servers and another stack is used with OE servers.

2. A *shared-stack* implementation, where you assign alternate port numbers to one set of the servers.

If you need to start both sets of servers, we recommend the separate-stack implementation, because this is the implementation that will be the most convenient for your client users.

In fact, the separate-stack implementation may be your only choice if you want to allow clients to use, for example, both the OE and the non-OE REXECD or RSHD servers. Most RSH and REXEC client implementations do not allow the end user to enter an alternate server port number, but will always send the requests to the well-known RSHD and REXECD server port numbers.

4.3 TCP/IP for MVS, Shared Stack

This configuration is based on a single TCP/IP for MVS stack that supports both native TCP/IP for MVS socket applications and OpenEdition socket applications. You may select this type of configuration if you do not want to start both a non-OE and an OE version of the same server programs. If, for example, you only want to use the Internet Connection Server for MVS in your OpenEdition environment, then there are no port number conflicts with non-OE applications and you can base your setup on a shared stack without any problems. If you, on the other hand, need to start both OE versions and non-OE versions of the same server programs, we recommend you choose a separate-stack configuration as described in 4.4, “TCP/IP for MVS, Separate Stacks” on page 92.

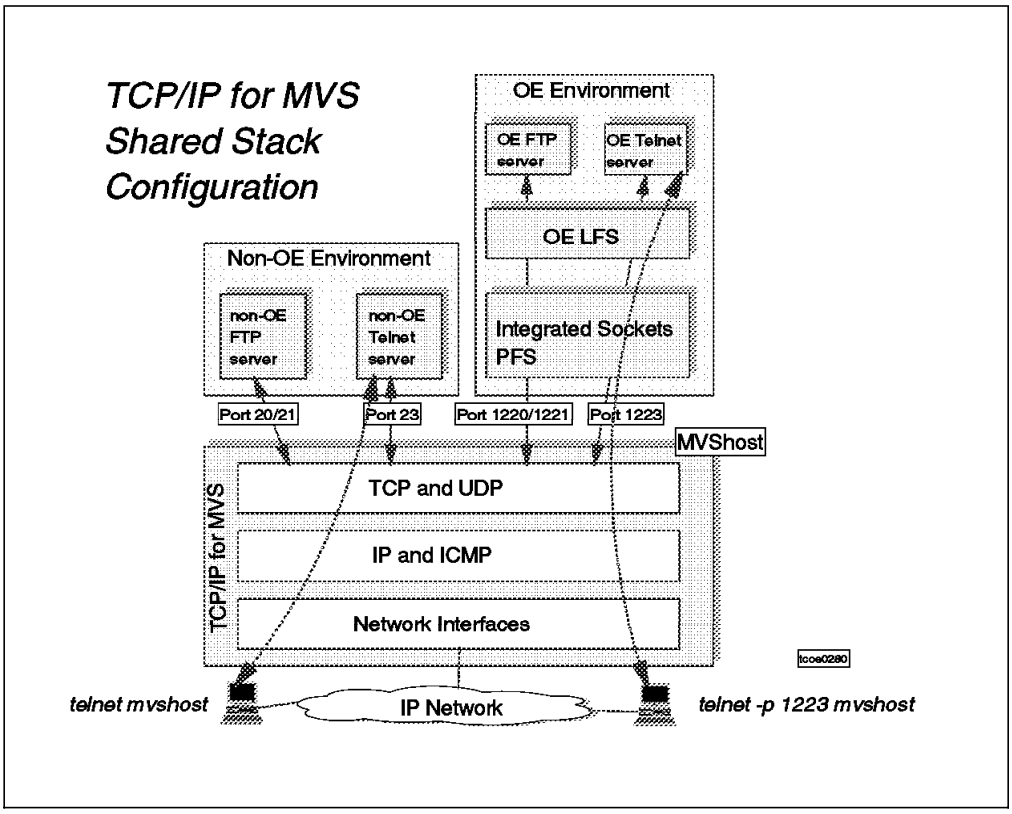


Figure 49. TCP/IP for MVS, Shared Stack Configuration

To set up an OpenEdition environment with TCP/IP for MVS as the single transport provider used for both standard TCP/IP applications and OE applications, the following steps must be performed:

1. Define a RACF user ID with an OMVS segment and assign it to the started task name of the TCP/IP system address space.
2. Customize SYS1.PARMLIB(BPXPRMxx) to use the integrated sockets physical file system.
3. Assign alternate port numbers to either the OE or the non-OE server applications.

4.3.1 RACF Definitions

A TCP/IP stack that is used as transport provider for OpenEdition needs both an MVS identity and an OpenEdition identity in terms of a UID. A transport provider stack needs superuser authority, which can be achieved by either a UID=0 or by assigning the TRUSTED attribute to the started task procedure name of the TCP/IP system address space.

The ITSO Raleigh convention is to use a started task procedure name that matches the associated started task user ID. In this example, the started task procedure name and user ID is T18ATCP.

```
listuser t18atcp

 USER=T18ATCP  1  NAME=TCP/IP OPENEDITION
  DEFAULT-GROUP=OMVSGRP  2
  ATTRIBUTES=NONE
  REVOKE DATE=NONE   RESUME DATE=NONE
  LAST-ACCESS=96.085/11:36:26
  CLASS AUTHORIZATIONS=NONE
  NO-INSTALLATION-DATA
  NO-MODEL-NAME
  LOGON ALLOWED   (DAYS)          (TIME)
  -----------------------------------------
  ANYDAY                          ANYTIME
   GROUP=OMVSGRP   AUTH=USE
     CONNECTS=    85  UACC=NONE     LAST-CONNECT=96.085/11:36:26
     CONNECT ATTRIBUTES=NONE
     REVOKE DATE=NONE   RESUME DATE=NONE
 SECURITY-LEVEL=NONE SPECIFIED
 CATEGORY-AUTHORIZATION
  NONE SPECIFIED
 SECURITY-LABEL=NONE SPECIFIED

 OMVS INFORMATION
 ----------------
 UID= 0000000000  3
 HOME= /
 PROGRAM= /bin/sh

rlist started t18atcp.* stdata

 STDATA INFORMATION
 ------------------
 USER= T18ATCP  1
 GROUP= OMVSGRP  2
 TRUSTED= NO   4
 PRIVILEGED= NO
 TRACE= NO
```

Figure 50. RACF Definitions for a Shared TCP/IP Stack

1 T18ATCP is the RACF started task user ID of the T18ATCP started task procedure.

2 OMVSGRP is the RACF group name of this TCP/IP for MVS stack.

3 This TCP/IP for MVS stack has an OMVS UID of 0 assigned.

4 Because we defined the T18ATCP user as a superuser, we do not need to assign any special RACF privileges to this started task. If we had defined the T18ATCP user ID with a non-zero UID, we would have had to assign the TRUSTED attribute to this started task in order for it to act as an OpenEdition transport provider.

4.3.2 SYS1.PARMLIB(BPXPRMxx)

If a single TCP/IP for MVS stack is used as transport provider for OpenEdition, we can use the integrated sockets physical file system.

To use the integrated sockets physical file system, modify the definitions for AF_INET in SYS1.PARMLIB(BPXPRMxx) according to the following sample:

```
/* AF_INET file system for sockets - TCP/IP for MVS        */
/* Integrated socket support - BPXTIINT                    */

FILESYSTYPE TYPE(INET)
            ENTRYPOINT(BPXTIINT)  1
NETWORK DOMAINNAME(AF_INET)  2
        DOMAINNUMBER(2)
        MAXSOCKETS(10000)  3
        TYPE(INET)
```

Figure 51. BPXPRMxx for a Shared TCP/IP Stack Using Integrated Sockets

1 BPXTIINT specifies integrated sockets with TCP/IP for MVS as transport provider.

2 AF_INET is the socket addressing family for this transport provider.

3 The number of MAXSOCKETS should be large enough to open new sockets for OpenEdition applications.

For a detailed description of parameters in SYS1.PARMLIB(BPXPRMxx), please refer to *MVS/ESA SP 5.2.2 Initialization and Tuning Reference*, SC28-1452.

4.3.3 Assign Port Numbers

If one single TCP/IP stack is used both for standard TCP/IP applications and OE applications, the TCP/IP for MVS PROFILE data set must contain port reservations for both sets of server applications. In this example, the TCP/IP non-OE servers will use the default port numbers, while alternate port numbers are assigned to the OE servers. The alternate port numbers must be in the range above 1024, in order to avoid interference with any standard port use.

```
PORT
     7 UDP T18AMISC  1          ; MISCSERVER
     7 TCP T18AMISC             ; MISCSERVER
     9 UDP T18AMISC             ; MISCSERVER
     9 TCP T18AMISC             ; MISCSERVER
    19 UDP T18AMISC             ; MISCSERVER
    19 TCP T18AMISC             ; MISCSERVER
    20 TCP T18AFTPC             ; FTP server data port
           NOAUTOLOG            ; Do not restart if stopped previously
    21 TCP T18AFTPC             ; FTP C server control port
    23 TCP INTCLIEN             ; Telnet server
    25 TCP T18ASMTP             ; SMTP server
    53 TCP T18ADNS              ; Name Server
    53 UDP T18ADNS              ; Name Server
    80 TCP OMVS  3              ; OE MVS Web Server
   111 UDP T18APORT             ; Portmapper Server
   111 TCP T18APORT             ; Portmapper Server
   125 UDP T18ALLBD             ; NCS Local Location Broker
   161 UDP T18ASNMD             ; Agent Port for SNMP Messages
   162 UDP T18ASNMQ             ; SNMP Client port for Trap's
   512 TCP T18ARXEC             ; Remote Execution
   514 TCP T18ARXEC             ; Remote Execution
   515 TCP T18ALPD              ; Remote Printer Server
   520 UDP T18AROUT             ; RouteD Server
   580 UDP T18ANCPR             ; NCPROUTE for RIP support on NCPs
  1211 TCP OMVS  2              ; OE ONC/RPC portmapper
  1211 UDP OMVS                 ; OE ONC/RPC portmapper
  1212 TCP OMVS                 ; OE Remote Execution
  1213 TCP OMVS                 ; OE Rlogin
  1214 TCP OMVS                 ; OE Remote Execution
  1220 TCP OMVS                 ; OE FTP Server data port
           DELAYACKS            ; Delay transmission acknowledgements
  1221 TCP OMVS                 ; OE FTP server control port
  1223 TCP OMVS                 ; OE Telnet server
  2049 UDP T18ANFS              ; NFS Server
  2049 UDP T18ANFS              ; NFS Server
```

Figure 52. PROFILE.TCPIP Extract For a Shared TCP/IP Stack Using Integrated Sockets

1 Port reservations for standard TCP/IP servers refer to the started task name of the respective server address space.

2 OMVS indicates that this port is reserved for OpenEdition. Please note that you do not specify the individual OE server name here, but just OMVS. The port is reserved for OpenEdition′s use, and what it is being used for inside the OpenEdition environment is, for most servers, managed via the /etc/services file contents.

3 If there is no corresponding non-OE version of a server, you can use the standard port number for your OE application, such as, in this example, the Internet Connection Server for MVS.

To access the OpenEdition FTPD server of this TCP/IP for MVS stack, a client has to specify port number 1221 with the FTP command. Otherwise, he or she would access the standard C-FTPD server.

```
ftp mvshost 1221
```

When your TCP/IP for MVS PROFILE data set assigns alternate port numbers to the OE applications, you need to update the OE resolver /etc/services file with the matching specifications.

```
sunrpc          1211/tcp                # onc/rpc portmapper
sunrpc          1211/udp                # onc/rps portmapper
exec            1212/tcp                # remote execution (rexec)
login           1213/tcp                # rlogin
shell           1214/tcp          cmd   # remote shell (rsh)
ftp             1221/tcp                # ftp
telnet          1223/tcp                # telnet
```

Figure 53. /etc/services with Alternate Port Numbers

Both InetD and the OE FTPD server will use a getservbyname() call to obtain the port numbers that you have assigned in the OE resolver /etc/services file to the individual services.

4.3.4 Resolver Configuration

As described in Chapter 3, “Setting Up the OE Resolver” on page 69, we recommend a separate set of resolver configuration files for the OE resolver, and another set of resolver configuration data sets for the TCP/IP for MVS resolver. See 3.7, “OE Resolver Configuration Summary” on page 80 for an overview of resolver configuration data.

4.4 TCP/IP for MVS, Separate Stacks

In this configuration, two TCP/IP stacks will be running on the same MVS/ESA system, but only one of them will be used as AF_INET transport provider for OpenEdition applications.

Choose this configuration if you have to support both an OE version and a non-OE version of the same servers, for example, both the OE telnet server and the non-OE telnet server.

This configuration is probably going to be an attractive configuration for most installations, and it is the configuration we recommend. It allows full convenient use of the non-OE server functions as well as full use of the OE server functions. Your users can use their client functions without having to remember alternate port numbers, they only have to remember two host names: one host name for your non-OE stack and another host name for your OE stack.

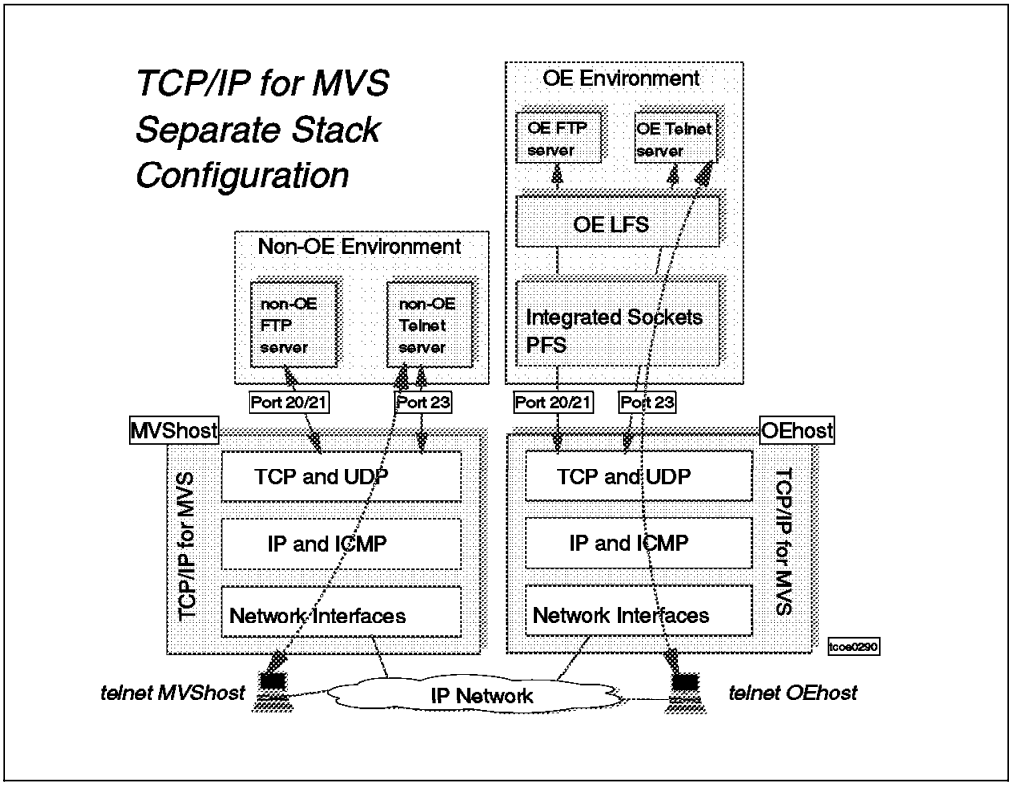


Figure 54. TCP/IP for MVS, Separate Stack Configuration

One stack is a TCP/IP for MVS stack containing all standard TCP/IP applications and servers without any connection to OpenEdition. The PROFILE data set of this TCP/IP stack must include the keyword NOOE to prevent this stack from establishing a connection to the OpenEdition integrated sockets physical file system. If you are already running a TCP/IP stack on your MVS system, no additional changes for this stack are required. For an initial setup of a standard TCP/IP environment, please follow the TCP/IP for MVS installation and customization guidelines. An example on how to set up such an environment is given in *TCP/IP V3R1 for MVS Implementation Guide*, GG24-3687.

To establish physical network connections from both your OE stack and your non-OE stack, you have two options available:

1. One of the TCP/IP stacks has a physical access to the IP network, for example, via an IBM 3172. The two TCP/IP stacks are connected using an IUCV link. The second TCP/IP stack uses this IUCV link and the physical connection of the partner stack for all network accesses.
2. Each of the two TCP/IP stacks has its own physical connection to the IP network. An IUCV link between the stacks is not required.

See Figure 55 on page 94 for an overview of these two alternative configurations.

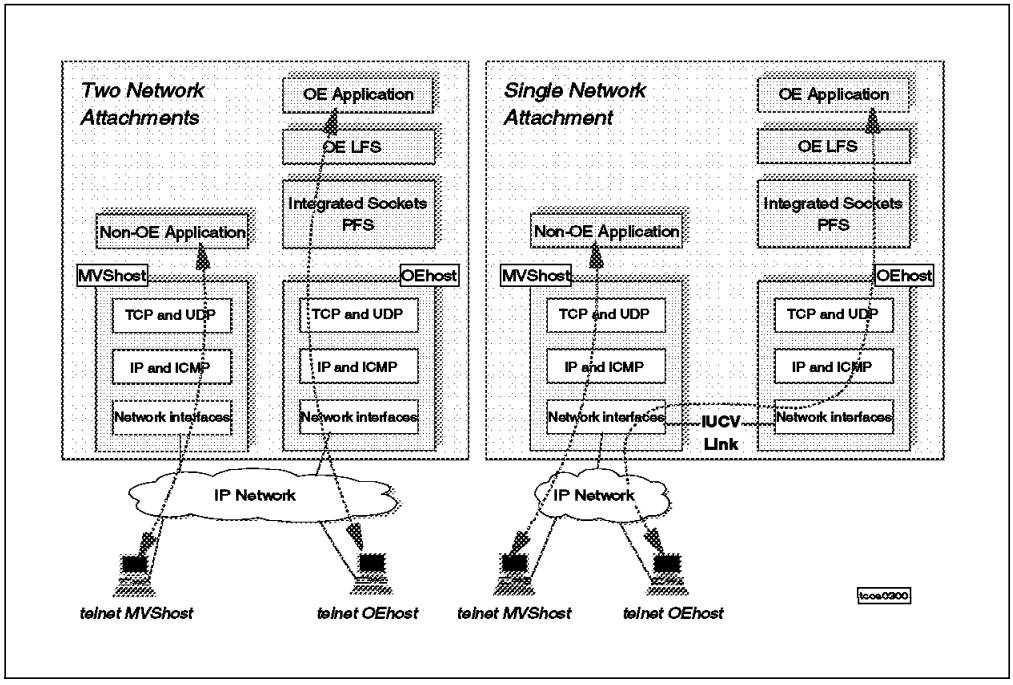


Figure 55. TCP/IP for MVS, Separate Stack Configuration, Network Attachment Options

The second TCP/IP stack will be used as single transport provider for OpenEdition. To allow establishment of an OpenEdition connection and use this second TCP/IP stack as transport provider for OE integrated sockets, the following actions are required:

1. Define a RACF user ID with an OMVS UID and assign it to the started task name of the TCP/IP for MVS system address space that is going to be used as your OpenEdition AF_INET transport provider.
2. Customize SYS1.PARMLIB(BPXPRMxx) to use the integrated sockets physical file system.
3. Optionally customize your TCP/IP for MVS PROFILE data sets with an IUCV link between the two TCP/IP for MVS stacks.

4.4.1 RACF Definitions

There is no requirement for the non-OE stack to have an OMVS UID. The stack is not connecting to OpenEdition and does not use any OpenEdition services.

The stack that connects to OpenEdition must have a valid OMVS UID, as described in 4.3, “TCP/IP for MVS, Shared Stack” on page 87.

```
listuser t18otcp

 USER=T18OTCP  1  NAME=T18O TCP/IP STACK
  DEFAULT-GROUP=OMVSGRP  2
  ATTRIBUTES=NONE
  REVOKE DATE=NONE   RESUME DATE=NONE
  LAST-ACCESS=96.085/11:42:21
  CLASS AUTHORIZATIONS=NONE
  NO-INSTALLATION-DATA
  NO-MODEL-NAME
  LOGON ALLOWED   (DAYS)          (TIME)
  ---------------------------------------------
  ANYDAY                          ANYTIME
   GROUP=OMVSGRP   AUTH=USE
     CONNECTS=   313  UACC=NONE
     CONNECT ATTRIBUTES=NONE
     REVOKE DATE=NONE   RESUME DATE=NONE
 SECURITY-LEVEL=NONE SPECIFIED
 CATEGORY-AUTHORIZATION
  NONE SPECIFIED
 SECURITY-LABEL=NONE SPECIFIED

 OMVS INFORMATION
 ----------------
 UID= 0000000000  3
 HOME= /
 PROGRAM= /bin/sh

rlist started t18otcp.* stdata

 STDATA INFORMATION
 ------------------
 USER= T18OTCP  1
 GROUP= OMVSGRP  2
 TRUSTED= NO  3
 PRIVILEGED= NO
 TRACE= NO
```

Figure 56. RACF Definitions for a Separate OE Stack

1 T18OTCP is the RACF started task ID of this TCP/IP for MVS stack.

2 OMVSGRP is the RACF group ID of this TCP/IP for MVS stack.

3 This TCP/IP for MVS stack has an OMVS UID of 0 assigned.

4 TRUSTED=NO, as the OMVS UID of this stack is 0.

4.4.2 SYS1.PARMLIB(BPXPRMxx) Definitions

Although two TCP/IP stacks are running on the same MVS system, only one of them is used as single transport provider for OpenEdition. We can use the integrated sockets physical file system.

Modify the definitions for AF_INET in SYS1.PARMLIB(BPXPRMxx) according to the following sample to use the integrated sockets physical file system.

```
/* AF_INET file system for sockets                          */
/* integrated socket support - BPXTIINT                     */

FILESYSTYPE TYPE(INET)
            ENTRYPOINT(BPXTIINT) 1
NETWORK DOMAINNAME(AF_INET) 2
        DOMAINNUMBER(2)
        MAXSOCKETS(10000) 3
        TYPE(INET)
```

Figure 57. BPXPRMxx for Separate OE Stack Using Integrated Sockets

1 BPXTIINT specifies integrated sockets with TCP/IP for MVS as transport provider.

2 AF_INET is the socket address type for this transport provider.

3 Parameter MAXSOCKETS should be large enough to open new sockets for OpenEdition applications.

4.4.3 PROFILE.TCPIP Customization

When separate TCP/IP stacks are used to run standard TCP/IP applications and OE applications, the PROFILE data set of the OpenEdition stack only includes definitions for the OE applications and any required non-OE servers on that stack, such as a RouteD server or maybe a domain name server.

```
PORT
    7 UDP  OMVS 1              ; OE Echo
    7 TCP  OMVS                ; OE Echo
    9 UDP  OMVS                ; OE Discard
    9 TCP  OMVS                ; OE Discard
   13 UDP  OMVS                ; OE Daytime
   13 TCP  OMVS                ; OE Daytime
   19 UDP  OMVS                ; OE Chargen
   19 TCP  OMVS                ; OE Chargen
   20 TCP  OMVS                ; OE FTP Server
            DELAYACKS          ; Delay transmission acknowledgements
   21 TCP  OMVS                ; OE FTP server control port
   23 TCP  OMVS                ; OE Telnet server
   37 TCP  OMVS                ; OE Timeserver
   37 UDP  OMVS                ; OE Timeserver
   53 TCP  T18ODNS 2           ; Caching-only name server
   53 UDP  T18ODNS 2           ; Caching-only name server
   80 TCP  OMVS                ; OE WEB server
   111 UDP OMVS                ; OE Portmapper Server
   111 TCP OMVS                ; OE Portmapper Server
   123 TCP OMVS                ; OE Web server ???
   512 TCP OMVS                ; OE Remote Execution Server
   513 TCP OMVS                ; OE Rlogin Server
   514 TCP OMVS                ; OE Remote Shell Server
   514 UDP OMVS 3              ; OE SyslogD Server
   520 UDP T18OROUT 2          ; RouteD Server
  8000 TCP OMVS                ; OE WEB server
```

Figure 58. PROFILE.TCPIP Extract for a Separate OE Stack

1 Keyword OMVS indicates that these ports are reserved for OpenEdition.

2 The only port numbers that are reserved for non-OE applications in this sample setup is the domain name server and the RouteD port numbers, which allow us to run these two non-OE servers on the OE stack. This can be done without conflicts with OE servers, because there currently is no domain name server or RouteD servers in the OpenEdition socket environment.

When you use the standard port numbers for your OpenEdition applications, you do not need to make any corrections to the OE resolver /etc/services file.

3 Please note that we reserve UDP port 514 to OMVS too. This port is used by the SyslogD server in OMVS to receive log messages from other SyslogD servers in your TCP/IP network.

The only modification you might want to make in the TCP/IP PROFILE data set of your non-OE stack is if you want to add an IUCV point-to-point link between your OE and your non-OE stack. See Figure 59 for the definitions in your non-OE stack, and see Figure 60 on page 98 for the matching definitions in your OE stack.

```
;
; This is the non-OE stack - T18ATCP
; *********************************
;
; Do not connect to OE
NOOE
;
; Device and Link definitions for IUCV link to the OE stack
;
DEVICE DEVTOO   IUCV XYZZY XYZZY T18OTCP A  1
LINK   LINKTOO  IUCV 0     DEVTOO

HOME
   192.168.254.2   LINKTOO     ; To T18OTCP - the OE stack

BSDROUTINGPARMS false
   LINKTOO   2000       0     255.255.255.0   192.168.254.1

START DEVTOO                    ; Start IUCV link to T18OTCP
```

Figure 59. TCPIP.PROFILE IUCV Link Definitions in a Non-OE Stack

1 The IUCV device statement must be coded with an A in one stack and a B in the other stack. The sequence with XYZZY XYZZY must be coded exactly as specified.

```
;
; This is the OE stack - T18OTCP
; ******************************
;
; Device and Link definitions for IUCV link to non-OE stack
;
DEVICE DEVTOA   IUCV XYZZY XYZZY T18ATCP B 1
LINK   LINKTOA  IUCV 0     DEVTOA

HOME
   192.168.254.1   LINKTOA     ; To T18ATCP - the non-OE stack

BSDROUTINGPARMS false
   LINKTOA   2000       0     255.255.255.0   192.168.254.2

START DEVTOA                    ; Start IUCV link to T18ATCP
```

Figure 60. TCPIP.PROFILE IUCV Link Definitions in an OE Stack

In this sample setup, the IUCV link definitions create a point-to-point link between the stacks using a private class C network 192.168.254.0 with the endpoint addresses 192.168.254.1 assigned to the OE stack and 192.168.254.2 to the non-OE stack.

4.4.4 Resolver Configuration

As described in Chapter 3, “Setting Up the OE Resolver” on page 69, we recommend a separate set of resolver configuration files for the OE resolver, another set of resolver configuration data sets for the OE TCP/IP for MVS resolver, and a third set of resolver configuration data sets for the non-OE TCP/IP for MVS stack. See 3.7, “OE Resolver Configuration Summary” on page 80 for an overview of resolver configuration data.

4.5 AnyNet MVS, Single Stack

If you do not have TCP/IP for MVS installed, but want to enable access to OpenEdition socket applications through an SNA network based on AnyNet nodes, you can choose this configuration. In an AnyNet MVS scenario you most likely do not have any concerns about supporting both OE and non-OE versions of your server programs because only a few server programs are able to work directly on top of the AnyNet MVS socket library. A separate stack implementation is therefore not very likely to be considered at all, and we do not describe the details for such a configuration in this document. If you do find yourself in a situation where you have to establish an AnyNet MVS separate stack configuration, the issues you face are the same as those you face with a separate stack TCP/IP for MVS configuration.

In this configuration, AnyNet MVS is the single transport provider for OpenEdition. You may have a TCP/IP for MVS system running all standard TCP/IP applications and servers without any connection to OpenEdition. The setup of that TCP/IP stack is completely independent of the OpenEdition environment.

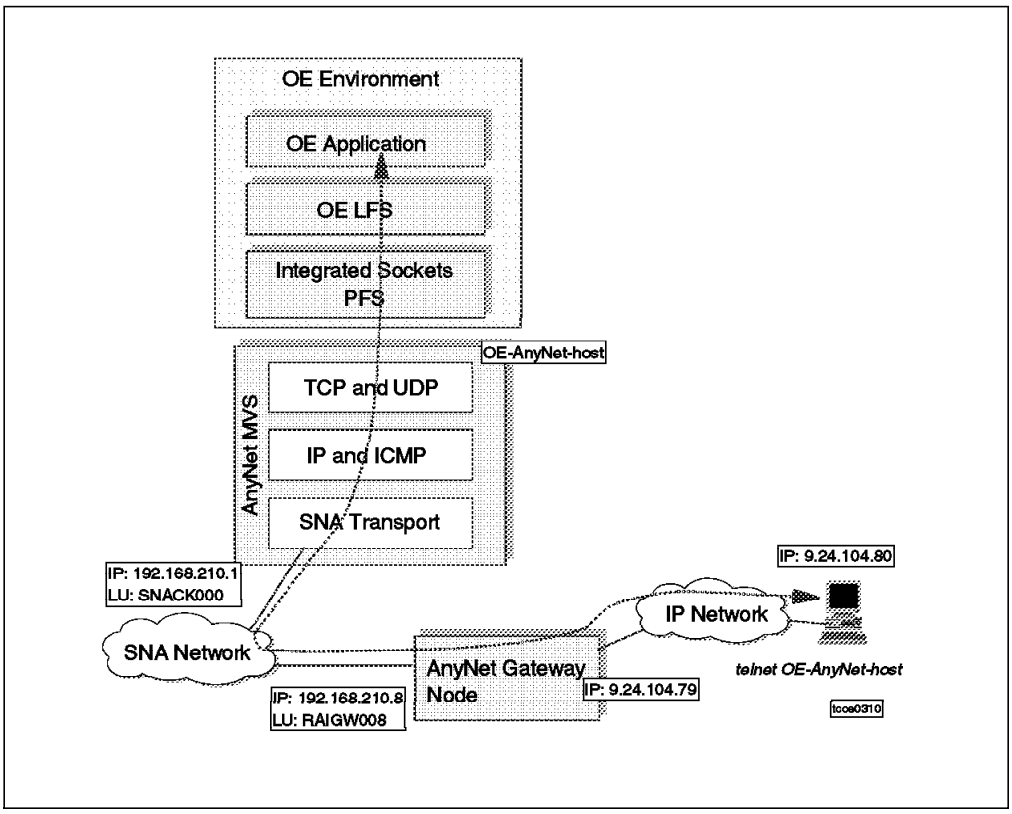


Figure 61. AnyNet MVS, Shared Stack Configuration

To use AnyNet MVS as transport provider for OpenEdition, the following actions are required:

1. Define a RACF user ID and OMVS UID for AnyNet MVS.
2. Customize SYS1.PARMLIB(BPXPRMxx) to use integrated sockets with AnyNet MVS.
3. Set up VTAM resource definitions for AnyNet MVS.
4. Customize the AnyNet MVS started task and configuration data sets.
5. Customize the AnyNet MVS initialization procedure.

4.5.1 RACF Definitions

The AnyNet stack used as transport provider for OpenEdition needs a RACF started task user ID with OpenEdition superuser authority, which can be achieved by either using a UID of zero or assigning the TRUSTED attribute to the AnyNet MVS started task procedure name.

In the following sample, an OE UID=0 is used for the AnyNet MVS transport provider stack.

```
listuser raisock

 USER=RAISOCK  NAME=ANYNET SA18 1
  DEFAULT-GROUP=OMVSGRP 2
  ATTRIBUTES=NONE
  REVOKE DATE=NONE   RESUME DATE=NONE
  LAST-ACCESS=96.085/11:37:31
  CLASS AUTHORIZATIONS=NONE
  NO-INSTALLATION-DATA
  NO-MODEL-NAME
  LOGON ALLOWED   (DAYS)          (TIME)
  -------------------------------------------
  ANYDAY                          ANYTIME
   GROUP=OMVSGRP   AUTH=USE
     CONNECTS=    77  UACC=NONE     LAST-CONNECT=96.085/11:37:31
     CONNECT ATTRIBUTES=NONE
     REVOKE DATE=NONE   RESUME DATE=NONE
 SECURITY-LEVEL=NONE SPECIFIED
 CATEGORY-AUTHORIZATION
  NONE SPECIFIED
 SECURITY-LABEL=NONE SPECIFIED

 OMVS INFORMATION
 ----------------
 UID= 0000000000 3
 HOME= /
 PROGRAM= /bin/sh

rlist started raisock.* stdata

 STDATA INFORMATION
 ------------------
 USER= RAISOCK 1
 GROUP= OMVSGRP 2
 TRUSTED= NO 4
 PRIVILEGED= NO
 TRACE= NO
```

Figure 62. RACF Definitions for Integrated Sockets AnyNet MVS

1 RAISOCK is the RACF started task user ID of this AnyNet MVS stack. The started task procedure name is RAISOCK.

2 OMVSGRP is the RACF group ID of this AnyNet MVS stack.

3 This AnyNet MVS stack has an OMVS UID of 0 assigned.

4 TRUSTED=NO, as the OMVS UID of this stack is 0.

4.5.2 SYS1.PARMLIB(BPXPRMxx) Definitions

If only AnyNet MVS is used as the transport provider for OpenEdition, you can use the integrated sockets physical file system.

Modify the definitions for AF_INET in SYS1.PARMLIB(BPXPRMxx) according to the following sample to use integrated sockets with AnyNet MVS:

```
/* AF_INET file system for sockets      */
/* integrated socket support - ISTOEPIT */

FILESYSTYPE TYPE(APPNSNA)  1
            ENTRYPOINT(ISTOEPIT)
NETWORK DOMAINNAME(AF_INET)  2
        DOMAINNUMBER(2)
        MAXSOCKETS(10000)  3
        TYPE(APPNSNA)  4
```

Figure 63. SYS1.PARMLIB(BPXPRMxx) for AnyNet MVS

1 TYPE(APPNSNA) and ENTRYPOINT(ISTOEPIT) specify AnyNet MVS as transport provider for OpenEdition.

2 AF_INET is the socket addressing family for this transport provider.

3 The number of MAXSOCKETS should be large enough to open new sockets for OpenEdition applications.

4.5.3 VTAM Resource Definitions for AnyNet

In order to use AnyNet MVS as transport provider for OpenEdition, you need some VTAM resource definitions:

- An APPL major node for the AnyNet MVS Sockets over SNA application is always required.
- If you do not use dynamic resource definitions in your VTAM environment, or if your AnyNet MVS node may initiate LU6.2 conversations with other AnyNet nodes, you also need to define your partner AnyNet nodes. In the sample setup in this context, the partner AnyNet node is a LAN-attached OS/2 workstation that is configured as an AnyNet gateway node. We include a sample switched major node definition for this workstation.

4.5.3.1 VTAM APPL Major Node

Before AnyNet MVS Sockets over SNA can be started and initialized, a VTAM application major node must be defined and activated.

Our sample major node is called RAIANYAP. Include this major node in your ATCCONxx VTAMLST member to ensure it is activated during VTAM startup.

```
*
* VTAM V4.3 - AnyNet Sockets over SNA - RAIANYAP Major Node
*
         VBUILD TYPE=APPL
*
SNACKOOO APPL  ACBNAME=SNACKOOO, 1
               APPC=YES, 2
               PARSESS=YES,
               DSESLIM=10,
               DMINWNL=5,
               DMINWNR=5,
               AUTOSES=2, 3
               AUTH=(ACQ,PASS),
               OPERCNOS=ALLOW,
               ATNLOSS=ALL,
               MODETAB=ISTINCLM
```

Figure 64. VTAM APPL Major Node Definition

1 ACBNAME must be equal to the LUNAME that is used in your SXMAP initialization job (see 4.5.5, “AnyNet MVS Initialization Procedure” on page 105).

2 APPC=YES is required because Sockets over SNA operates as an LU6.2 application that uses the APPCCMD VTAM interface.

3 Allow automatic session initialization with an AnyNet partner node.

4.5.3.2 Partner AnyNet Node Definition

When using dynamic resource definitions in your VTAM configuration, you do not need to define the AnyNet/2 resources in advance if the LU6.2 sessions are initiated from the AnyNet/2 node. If your LU6.2 sessions are initiated from VTAM to the AnyNet/2 node, you need to define your AnyNet/2 resources in advance in order for VTAM to know which path to use to contact the AnyNet/2 node. The example shown in Figure 65 on page 103 can be used if your AnyNet/2 node is a LAN node that is attached through an NCP. Almost identical definitions could be used if your AnyNet/2 node were attached through a 3172 SNA gateway.

```
************************************************************************
*                                                                      *
*         VTAM SWITCHED MAJOR NODE FOR WTR05322                        *
*                                                                      *
************************************************************************
RAI05322 VBUILD MAXGRP=1,                                             X
               MAXNO=3,                                               X
               TYPE=SWNET
*
RAI5322P PU    ADDR=13,                                               X
               IDBLK=05D,                                             X
               IDNUM=05322,                                           X
               CPNAME=WTR05322, 1                                     X
               DISCNT=NO,                                             X
               ISTATUS=ACTIVE,                                        X
               MAXDATA=1033,                                          X
               MAXPATH=4,                                             X
               PACING=0,                                              X
               PUTYPE=2,                                              X
               MODETAB=ISTINCLM, 2                                    X
               USSTAB=US327X,                                         X
               VPACING=0
*
RAI5322X PATH  DIALNO=0004400052005322, 3                             X
               GRPNM=EG24P00,                                         X
               GID=1,                                                 X
               PID=2,                                                 X
               USE=YES
*
RAIGW008 LU    LOCADDR=0,DLOGMOD=SNACKETS  4
RAI5322A LU    LOCADDR=2,DLOGMOD=SNX32703  5
RAI5322B LU    LOCADDR=3,DLOGMOD=SNX32703
RAI5322C LU    LOCADDR=4,DLOGMOD=SNX32703
RAI5322D LU    LOCADDR=5,DLOGMOD=SNX32703
```

Figure 65. VTAM Switched Major Node Definition for Subarea Network

1 The value for CPNAME identifies the control point of the AnyNet Sockets over SNA workstation (local node definition). Independent LU6.2 nodes are required for Sockets over SNA.

2 The default AnyNet mode table entry is SNACKETS, which is part of the default logon mode table ISTINCLM. This table is shipped in source form in SYS1.SAMPLIB. If you maintain your own logon mode tables, you can copy the SNACKETS entry from ISTINCLM into your own table and reassemble it.

3 The PATH statement identifies how a VTAM subarea node can reach the workstation in case session initiation takes place from VTAM. In this example, the AnyNet/2 node can be reached through a token-ring adapter in an NCP. The TIC to use is identified by the NCP group label EG24P000 and the AnyNet/2 LAN hardware address (MAC address) is 400052005322.

4 The RAIGW008 LU is the independent LU6.2 node that is used for the sessions between the AnyNet/2 gateway node and our AnyNet MVS node.

5 The four dependent LUs that are defined in this sample switched major node are not used for Sockets over SNA communication, but for plain 3270 emulation from the OS/2 workstation.

If your AnyNet partner nodes are attached via leased lines to a subarea NCP node, you need similar non-switched definitions in your NCP deck.

4.5.4 AnyNet MVS Started Task and Configuration Data Sets

When the VTAM application major node for Sockets over SNA is active, AnyNet MVS Sockets over SNA may be started with the following JCL procedure:

```
//RAISOCK  PROC
//*
//* AnyNet MVS for OpenEdition use
//*
//ANYNET   EXEC PGM=ISTSKDMN,REGION=0M,TIME=1440 1
//STEPLIB  DD DSN=SYS1.VTAMLIB,DISP=SHR
//         DD DSN=CEE.V1R5M0.SCEERUN,DISP=SHR 2
//SYSUDUMP DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//CEEDUMP  DD SYSOUT=* 3
//ENVVAR   DD DSN=ANYNET.MVS(ENVVAR),DISP=SHR 4
//LOGMSGS  DD SYSOUT=*
//SYSOUT   DD SYSOUT=* 5
//*        PEND
```

Figure 66. AnyNet MVS JCL Procedure

1 Minimum recommended Region Size is 8MB. By specifying zero MB (0M), we allow AnyNet MVS to use as much virtual storage as needed.

2 Language Environment or C/370 run-time library must be either in LINKLST concatenation or specified as STEPLIB.

3 A CEEDUMP data set is required for Language Environment.

4 ENVVAR specifies the AnyNet MVS configuration data set. If no ENVVAR data set is specified, AnyNet MVS Sockets over SNA will be started with default values.

5 SYSOUT data set is required for run-time messages from Language Environment.

AnyNet MVS Sockets over SNA can be started with default environment settings. We recommend, however, that you use an ENVVAR data set and explicitly specify all parameters, even if the default values are used. Thus the AnyNet MVS ENVVAR data set will always reflect your active parameter settings.

```
ADDRINFO='TCPIP.ANYNET.HOSTS.ADDRINFO'
SITEINFO='TCPIP.ANYNET.HOSTS.SITEINFO'
DNS_XLATE_TABLE='TCPIP.ANYNET.STANDARD.TCPXLBIN'
HOSTS_FILE_FORMAT=MVSTCP 1
HOSTNAME=MVS18AN 2
ETC_PROTOCOLS='TCPIP.ANYNET.ETC.PROTO'
ETC_RESOLV='TCPIP.ANYNET.RESOLVER'
ETC_SERVICES='TCPIP.ANYNET.ETC.SERVICES'
GROUP_NAME=SNACKETS
SXMODE_DEFAULT=SNACKETS 3
OPEN_EDITION=YES 4
```

Figure 67. AnyNet MVS ENVVAR Data Set

1 The AnyNet MVS resolver can use two different formats for the hosts file:

- HOSTS_FILE_FORMAT=BSD

The AnyNet MVS resolver will access the data sets that are pointed to by the ETC_HOSTS and ETC_NETWORKS keywords and will read them according to the rules for BSD formatted hosts files.

- HOSTS_FILE_FORMAT=MVSTCP

  The AnyNet MVS resolver will access the data sets that are pointed to by the ADDRINFO and SITEINFO keywords and read them according to the rules for hosts files that have been created with the TCP/IP for MVS MAKESITE utility program.

In this sample setup, all HOSTS data sets are formatted by TCP/IP for MVS. This is usually valid if both AnyNet MVS and TCP/IP for MVS are running on the same MVS system. MVSTCP is required when you use the integrated sockets physical file system with OpenEdition. If you use the converged sockets physical file system, both formats are valid. You may have to consider this if you want to use AnyNet MVS as the single AF_INET transport provider in an environment where you do not have TCP/IP for MVS installed. In that situation you do not have TCP/IP for MVS MAKESITE utility available and cannot create the MVSTCP formatted hosts file. In that situation, you can use the converged sockets physical file system instead, but just with AnyNet MVS as a single AF_INET transport provider.

2 The HOSTNAME identifies the identity of the AnyNet MVS TCP/IP stack. Your name server should map this name to the IP address that is mapped to the AnyNet MVS LU name (see 4.5.5, “AnyNet MVS Initialization Procedure”).

3 Default DLOGMOD for AnyNet Sockets over SNA is SNACKETS on all AnyNet platforms.

4 Establish connection to OpenEdition.

The RESOLVER_CONFIG data set contains definitions for the domain name of the Sockets over SNA IP network and the IP address of the domain name server. This data set is not used by OpenEdition socket applications. It is only used by native AnyNet MVS socket applications. OpenEdition socket applications use the OE resolver configuration data set or file.

```
domain itso.ral.ibm.com
nameserver  9.24.104.108
```

Figure 68. AnyNet MVS RESOLVER.CONFIG Data Set

4.5.5 AnyNet MVS Initialization Procedure

After the AnyNet MVS Sockets over SNA address space has been started, the IP-LU mapping table must be initialized and the sna0 interface defined and set up. The AnyNet MVS utilities ISTSKMAP, ISTSKRTE, and ISTSKIFC can be used either interactively or in a batch job for these tasks.

In the following example, we use a batch TSO started task to execute the AnyNet commands to update the AnyNet MVS configuration. The started task name in our environment is RAISOCKI, as shown in Figure 69 on page 106.

```
//RAISOCKI PROC MEMBER=RAIANYI
//*********************************************************************
//*
//* Initialize AnyNet MVS on MVS18
//*
//* Initialization TSO commands are in member RAIANYI in
//* RISC.VTAMLST
//*
//*********************************************************************
//*
//ANYNETI  EXEC PGM=IKJEFT01,DYNAMNBR=20
//STEPLIB  DD DSN=SYS1.VTAMLIB,DISP=SHR
//         DD DSN=SYS1.SISTLMD1,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSTSIN  DD DSN=RISC.VTAMLST(&MEMBER.),DISP=SHR
//SYSIN    DD DUMMY
```

Figure 69. AnyNet MVS Initialization Procedure

If you are using an operations automation package, for example, AOC/MVS Automated Operations Control from IBM, you can establish message automation based on the following message from AnyNet MVS:

```
ISU1501I SOCKETS-OVER-SNA RAISOCK INITIALIZATION COMPLETE FOR V4R3
```

When this message is received, your automation package can start the AnyNet MVS initialization job, in our sample setup, the started task RAISOCKI.

The input file to the AnyNet MVS initialization procedure is in our setup a set of TSO commands as shown in Figure 70 on page 107.

```
/*********************************************************************/
/*                                                                   */
/* RISC.VTAMLST(RAIANYI) - Applies to RAISOCK on MVS18               */
/*                                                                   */
/* This member specifies the AnyNet MVS initialization               */
/* commands.  These commands must be executed after                  */
/* AnyNet MVS has been started.                                      */
/*                                                                   */
/*********************************************************************/
/*                                                                   */
/* First we map IP addresses in the AnyNet network to SNA            */
/* LU6.2 names.                                                      */
/*                                                                   */
/*********************************************************************/
istskmap flush
istskmap add 192.168.210.1 255.255.255.255 USIBMRA SNACK000 1
istskmap add 192.168.210.9 255.255.255.255 USIBMSC WTR05140
istskmap add 192.168.210.8 255.255.255.255 USIBMSC RAIGW008 2
istskmap add 192.168.210.2 255.255.255.255 USIBMRA SX221Q02
istskmap add 192.168.210.3 255.255.255.255 USIBMRA SX221Q03
istskmap get
/*********************************************************************/
/*                                                                   */
/* We add IP address 192.168.210.1 as the HOME IP address of         */
/* the AnyNet SNA0 interface.                                        */
/*                                                                   */
/*********************************************************************/
istskifc sna0 192.168.210.1 3
/*********************************************************************/
/*                                                                   */
/* We add a route to the 9.24.104.0 subnet via our OS/2              */
/* AnyNet gateway and another route to the 192.168.221.0 network     */
/* via a 2217 AnyNet gateway.                                        */
/*                                                                   */
/*********************************************************************/
istskrte add net 9.24.104 192.168.210.8 2 4
istskrte add net 192.168.221 192.168.210.3 2
/*********************************************************************/
/*                                                                   */
/* The netstat -r command displays the AnyNet MVS routing table      */
/*                                                                   */
/*********************************************************************/
istsknst -r
```

Figure 70. AnyNet MVS Initialization TSO Commands

Due to the relatively simple network setup in our sample environment, we chose to use explicit LU name to IP address mapping rules.

1 This command explicitly maps IP address 192.168.210.1 to LU name SNACK000. This is the definition for the AnyNet MVS host and the LU name therefore has to match the ACBNAME on the VTAM APPL major node definition.

2 This command explicitly maps the AnyNet/2 workstation IP address 192.168.210.8 to LU name RAIGW008. The LU name matches one of the independent LU6.2 names that have been defined on the AnyNet/2 gateway node.

3 This command sets the home IP address of the AnyNet MVS sna0 interface to 192.168.210.1.

4 With this command, routing information for workstations in the IP network is added. These workstations have no AnyNet Sockets over SNA running, but use the AnyNet/2 gateway node workstation instead. The ISTSKRTE route update utility allows you to modify the AnyNet MVS routing table. The 9.24.104 route is a subnet route to the 9.24.104.0 subnet of the class A 9.0.0.0 network subnetted with a mask of 255.255.255.0. The second route added is a network route to the class C 192.168.221.0 network.

For a detailed description of available commands, please refer to the *VTAM AnyNet Feature: Guide to Sockets over SNA*, SC31-6559.

The RAISOCKI task should end with a zero return code. Using the sample above, the following messages indicate a successful completion:

```
>  flush
>  add 192.168.210.1 255.255.255.255 usibmra snackOOO
>  add 192.168.210.9 255.255.255.255 usibmsc wtr05140
>  add 192.168.210.8 255.255.255.255 usibmsc raigwOO8
>  add 192.168.210.2 255.255.255.255 usibmra sx221q02
>  add 192.168.210.3 255.255.255.255 usibmra sx221q03
>  get

Address         Mask            Network Name    LU Template
-------------------------------------------------------------
192.168.210.3   FFFFFFFF        USIBMRA         SX221QO3
192.168.210.2   FFFFFFFF        USIBMRA         SX221QO2
192.168.210.8   FFFFFFFF        USIBMSC         RAIGWOO8
192.168.210.9   FFFFFFFF        USIBMSC         WTRO5140
192.168.210.1   FFFFFFFF        USIBMRA         SNACKOOO

add net 9.24.104: gateway 192.168.210.8
add net 192.168.221: gateway 192.168.210.3

   destination            gateway     refcnt        use  flags    intrf
    9.24.104.0      192.168.210.8       0             0    U      sna0
 192.168.210.0      192.168.210.1       0             0    U      sna0
 192.168.221.0      192.168.210.3       0             0    U      sna0
```

Figure 71. RAISOCKI Successful Job Completion

4.6 TCP/IP for MVS and AnyNet MVS As Transport Providers

If your network consists of both an IP section and an AnyNet section, you may want to allow access to your OpenEdition environment from both sections of your network concurrently. To do so, you can customize both TCP/IP for MVS and AnyNet MVS as AF_INET transport providers for OpenEdition, based on the OpenEdition converged sockets physical file system.

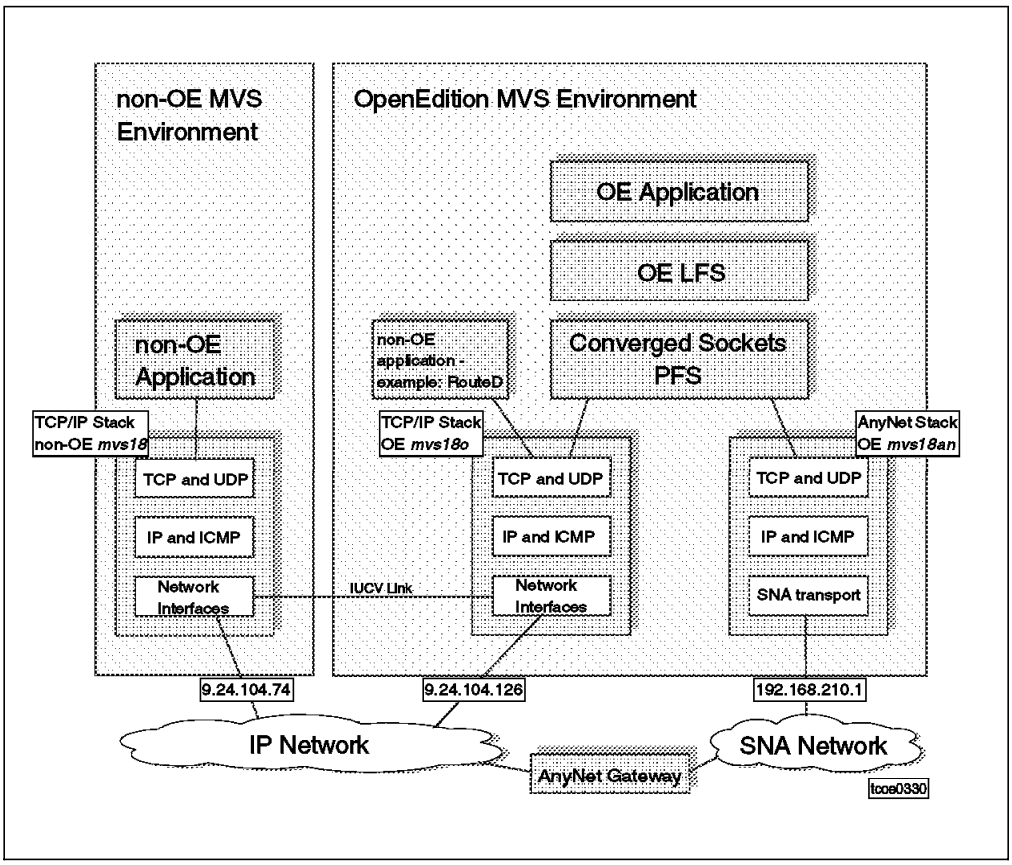


Figure 72. TCP/IP for MVS and AnyNet MVS as AF_INET Transport Providers

The configuration shown in Figure 72 consists of a non-OE TCP/IP for MVS stack, an OE TCP/IP for MVS stack and an AnyNet MVS stack. The TCP/IP for MVS OE stack and the AnyNet MVS stack both connect to the converged sockets physical file system as AF_INET transport providers.

The OE TCP/IP for MVS stack has to be configured according to the description in 4.4, “TCP/IP for MVS, Separate Stacks” on page 92, and the AnyNet MVS stack has to be configured according to the description in 4.5, “AnyNet MVS, Single Stack” on page 98. The only differences from the previously described configurations are:

- In this configuration where we work with more AF_INET transport providers, we have to use the converged sockets physical file system instead of the integrated sockets physical file system.
- Because we use the converged sockets physical file system, we need to reserve a number of ephemeral port numbers to be exclusively managed by OpenEdition. The same set of port numbers must be reserved in:
  1. BPXPRMxx
  2. TCP/IP for MVS PROFILE data set
  3. AnyNet MVS ENVVAR data set

4.6.1 SYS1.PARMLIB(BPXPRMxx) Definitions for Converged Sockets

For a detailed description of parameters in SYS1.PARMLIB(BPXPRMxx), please refer to *MVS/ESA Initialization and Tuning Reference*, SC28-1452.

```
/* AF_INET file system for sockets      */
/* Converged socket support - BPXTCINT  */

FILESYSTYPE TYPE(CINET)
            ENTRYPOINT(BPXTCINT) 1
NETWORK DOMAINNAME(AF_INET)
        DOMAINNUMBER(2)
        MAXSOCKETS(10000) 2
        TYPE(CINET)
        INADDRANYPORT(10000) 3
        INADDRANYCOUNT(2000)
SUBFILESYSTYPE NAME(T18OTCP) 4
               TYPE(CINET)
               ENTRYPOINT(BPXTIINT) 5
               DEFAULT 6
SUBFILESYSTYPE NAME(RAISOCK) 4
               TYPE(CINET)
               ENTRYPOINT(ISTOEPIT) 7
SUBFILESYSTYPE NAME(LINET)
               TYPE(CINET)
               ENTRYPOINT(BPXTLINT) 8
```

Figure 73. SYS1.PARMLIB(BPXPRMxx) for Converged Sockets, TCP/IP for MVS, AnyNet MVS

1 BPXTCINT specifies the use of converged sockets.

2 The number of MAXSOCKETS should be large enough to open new sockets for OpenEdition applications.

3 INADDRANYPORT and INADDRANYCOUNT specify the first ephemeral port number and the range of ports for OpenEdition. These values have to match the PORTRANGE definitions in your TCP/IP for MVS PROFILE data set and in the AnyNet MVS ENVVAR data set.

4 A transport provider stack for converged sockets is specified with a SUBFILESYSTYPE statement. The NAME keyword contains the RACF started task user ID of this transport provider stack (T18OTCP).

Note: Please note that the NAME keyword does *not* specify the started task name, but the started task RACF user ID of the transport provider address spaces.

5 BPXTIINT identifies a TCP/IP for MVS stack.

6 Keyword DEFAULT specifies which transport provider stack is to be used as the default stack for OpenEdition. If DEFAULT is not specified, the first stack will be used as the default stack. The sequence of SUBFILESYSTYPE statements is arbitrary if one stack is identified with the keyword DEFAULT.

7 ISTOEPIT identifies an AnyNet MVS stack.

8 A SUBFILESYSTYPE statement with ENTRYPOINT(BPXTLINT) is required for the optional local INET support. The local INET physical file system is used to enhance performance when two socket application programs in the OpenEdition

environment use AF_INET sockets for local socket communication. If an OpenEdition client program connects to a loopback IP address, the data transport between the local client and the local server does not flow through one of the AF_INET transport providers, but loops back inside the local INET physical file system. A loopback address is predefined as IP address 127.0.0.0.

4.6.2 PROFILE.TCPIP Customization for Converged Sockets

In addition to assigning port numbers to servers, you also need to reserve the same range of ephemeral port numbers that was reserved on the NETWORK statement in BPXPRMxx.

```
PORT
    7 UDP  OMVS  1                ; OE Echo
    7 TCP  OMVS                   ; OE Echo
    9 UDP  OMVS                   ; OE Discard
    9 TCP  OMVS                   ; OE Discard
   13 UDP  OMVS                   ; OE Daytime
   13 TCP  OMVS                   ; OE Daytime
   19 UDP  OMVS                   ; OE Chargen
   19 TCP  OMVS                   ; OE Chargen
   20 TCP  OMVS                   ; OE FTP Server
           DELAYACKS              ; Delay transmission acknowledgements
   21 TCP  OMVS                   ; OE FTP server control port
   23 TCP  OMVS                   ; OE Telnet server
   37 TCP  OMVS                   ; OE Timeserver
   37 UDP  OMVS                   ; OE Timeserver
   53 UDP  T18ODNS                ; Name server (caching-only)
   111 UDP OMVS                   ; OE Portmapper Server
   111 TCP OMVS                   ; OE Portmapper Server
   512 TCP OMVS                   ; OE Remote Execution
   513 TCP OMVS                   ; OE Rlogin
   514 TCP OMVS                   ; OE Remote Execution
   514 UDP OMVS                   ; OE SyslogD Server
   520 UDP T18OROUT               ; RouteD Server

PORTRANGE 10000 2000 TCP OMVS     ; TCP 10000 - 11999  2
PORTRANGE 10000 2000 UDP OMVS     ; UDP 10000 - 11999
```

Figure 74. PROFILE.TCPIP for the Default TCP/IP Stack

1 OMVS indicates that this port is reserved for OpenEdition.

Note: If you use two TCP/IP for MVS stacks as transport providers for OpenEdition, you *must* make the same port reservations for OMVS in both stacks. If one of the two stacks reserve a port number for OMVS and the other stack reserves the same port number for a non-OE application, the OpenEdition application will not be able to start.

2 The PORTRANGE statements reserve a range of ephemeral TCP and UDP ports for OpenEdition. In this example, ports 10000 to 11999 are reserved. The range must match the INADDRANYPORT and INADDRANYCOUNT in your BPXPRMxx member.

4.6.3 AnyNet MVS Definitions for Converged Sockets

The only change to your AnyNet MVS ENVVAR data set for converged sockets support is the addition of the OE_INADDRANY_PORT and IN_ADDRANY_COUNT keywords.

```
ADDRINFO=' TCPIP.ANYNET.HOSTS.ADDRINFO'
SITEINFO=' TCPIP.ANYNET.HOSTS.SITEINFO'
DNS_XLATE_TABLE=' TCPIP.ANYNET.STANDARD.TCPXLBIN'
HOSTS_FILE_FORMAT=MVSTCP
HOSTNAME=MVS18AN
ETC_PROTOCOLS=' TCPIP.ANYNET.ETC.PROTO'
ETC_RESOLV=' TCPIP.ANYNET.RESOLVER'
ETC_SERVICES=' TCPIP.ANYNET.ETC.SERVICES'
GROUP_NAME=SNACKETS
SXMODE_DEFAULT=SNACKETS
OPEN_EDITION=YES
OE_INADDRANY_COUNT=2000  1
OE_INADDRANY_PORT=10000
```

Figure 75. AnyNet MVS ENVVAR Data Set for Converged Sockets

1 These definitions must again match the corresponding definitions in your BPXPRMxx member.

4.7 AnyNet/2 Setup

AnyNet MVS Sockets over SNA needs a partner AnyNet node to be used as transport provider with OpenEdition. AnyNet products are available on several platforms. In our configuration, we used AnyNet/2 as the partner node to AnyNet MVS.

You can customize the AnyNet/2 workstation either as an access node or as a gateway node. Setting up AnyNet/2 as an access node will create a connection between AnyNet MVS and AnyNet/2 that can only be used by socket applications running on the AnyNet MVS host and this workstation.

If AnyNet/2 is configured as gateway node, other workstations in the IP network can use Sockets over SNA connections via this gateway node without having an AnyNet product installed. MPTS, TCP/IP for OS/2 and AnyNet/2 must be installed on the workstation prior to setting up and starting an AnyNet/2 node. You may want to back up your current workstation configuration before setting it up as an AnyNet/2 node.

To start configuring an AnyNet/2 node, select the **Communication Manager Setup** in the Communications Manager folder. Select **Setup** and open a configuration.

On the Communications Manager Configuration Definition panel, click on **Options** on the action bar and select the **Configure any Profile or Feature** option. Scroll down the list of available features to the SNA section.

First, select **SNA local node characteristics** and note both the network ID and the local node name of your workstation as displayed on the panel. The local node name must match the CPNAME keyword in your optional VTAM resource definitions (see Figure 65 on page 103). The network ID must match your IP to LU name mapping rules in AnyNet MVS (see Figure 70 on page 107, istskmap for IP address 192.168.210.8)

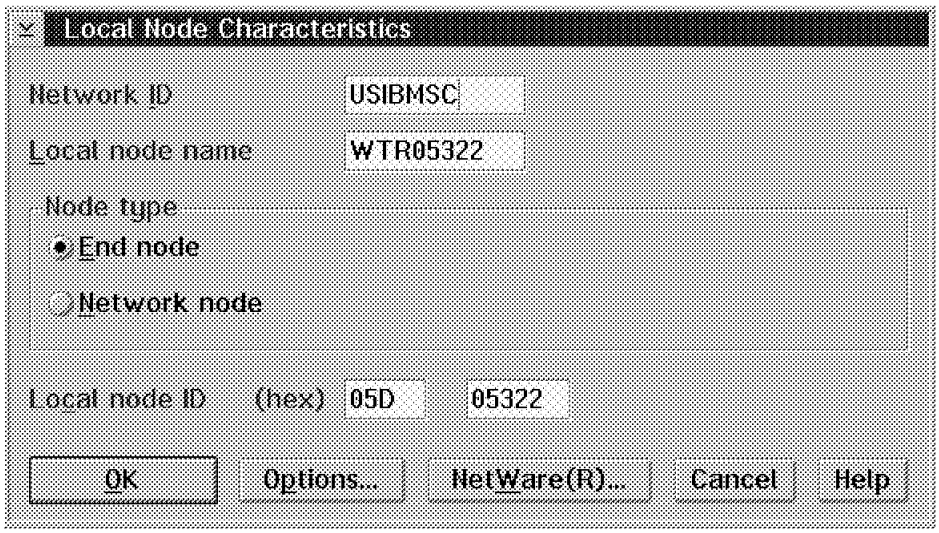


Figure 76. Verify Local Node Characteristics

Confirm the local node definitions by clicking on **OK**.

Next, select **SNA connections** from the configurable features list. If your VTAM is configured as an APPN node, you must create a connection to a network node.

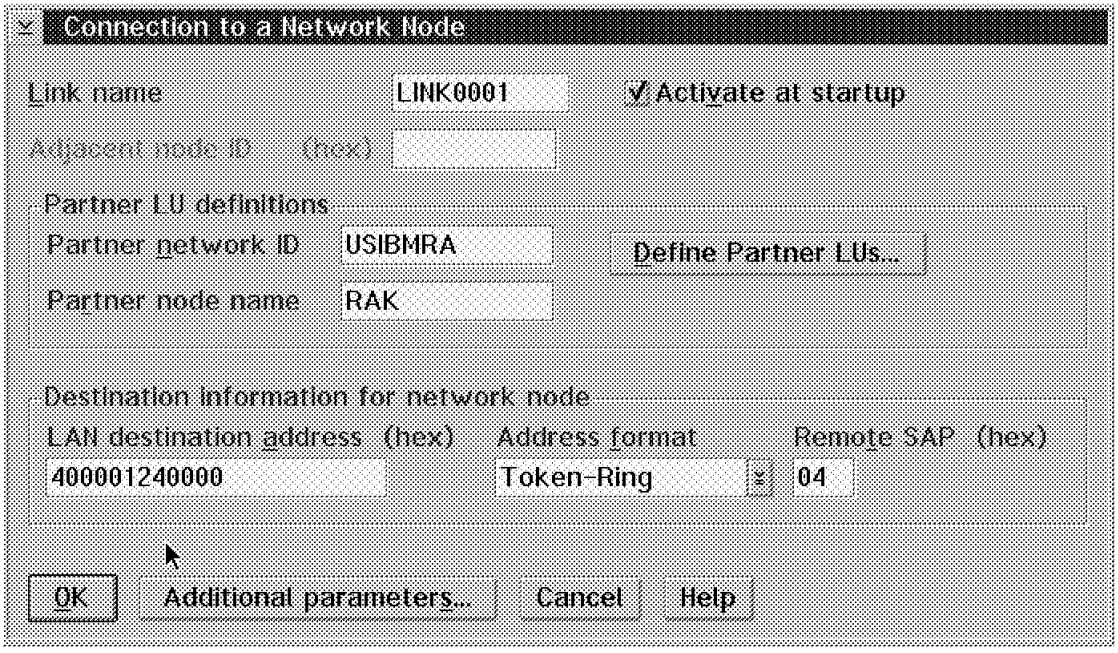


Figure 77. Define a Connection to a Network Node

The partner network ID and partner node name identifies your VTAM network node CPNAME. In this example, the VTAM network node RAK resides in network USIBMRA. This network node is reached via an NCP with a token-ring adapter that has a MAC address of 400001240000.

Select **Additional parameters** and click on **Preferred Server** in the next panel. Confirm your settings by selecting **OK**.

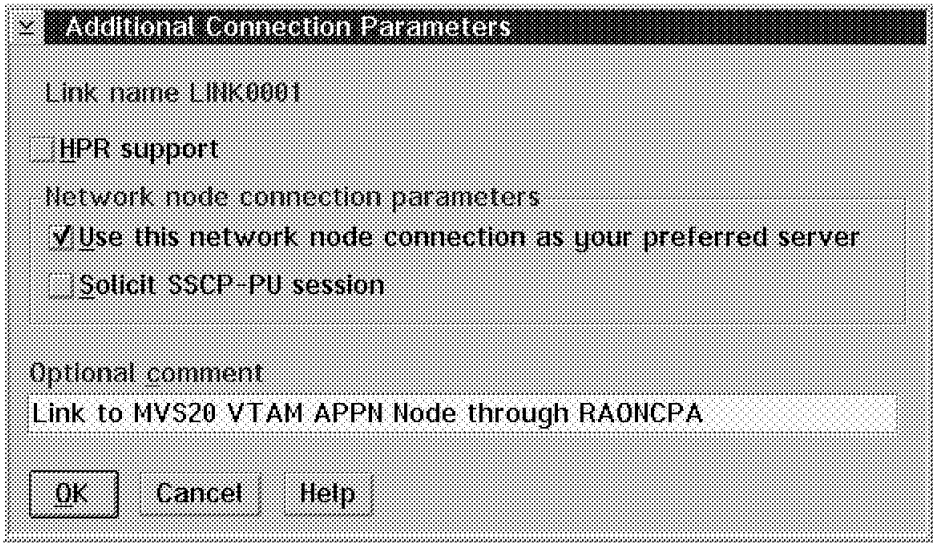


Figure 78. Define Additional Network Node Parameters

If your VTAM is configured as a subarea node, you must create a connection to a host instead of a connection to a network node.

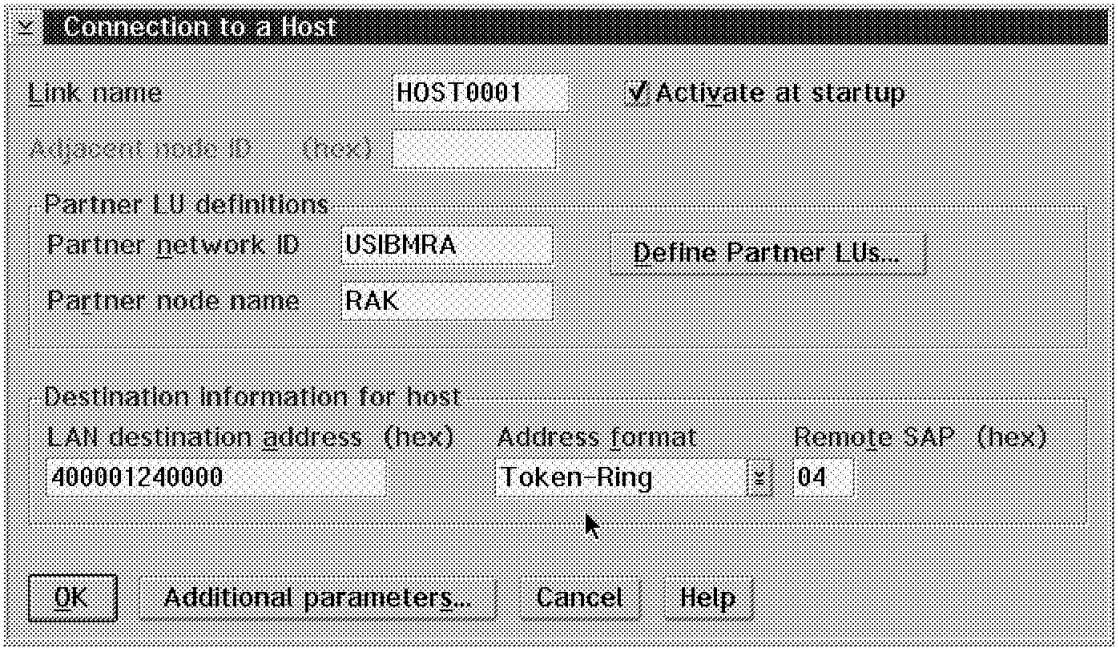


Figure 79. Define a Host Connection

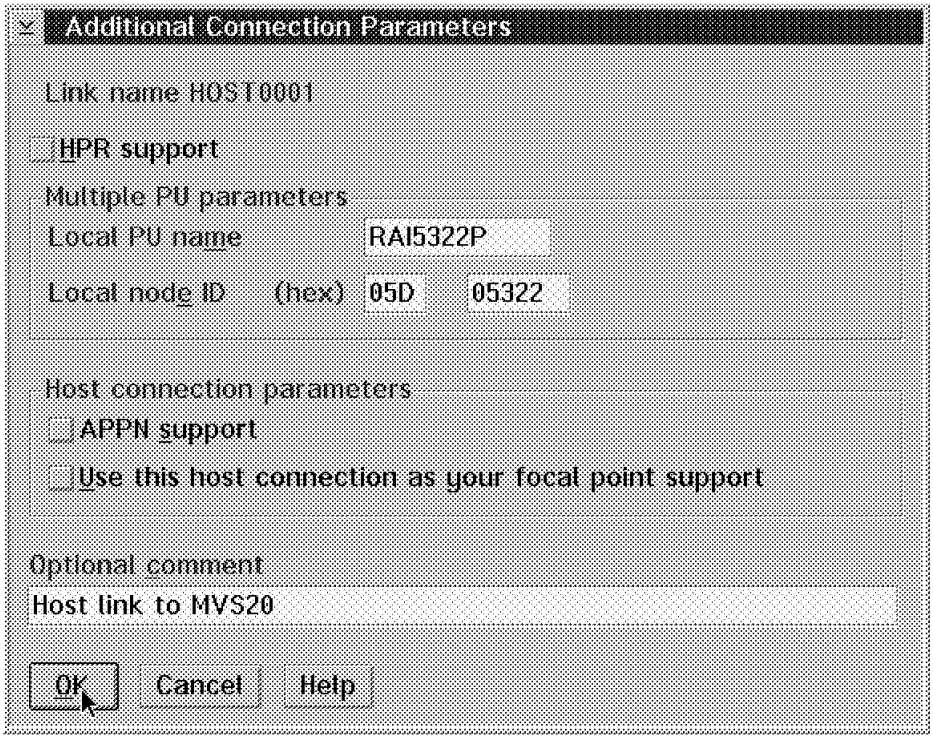


Figure 80. Set Additional Host Connection Parameters

You can either use the local CPNAME as your Sockets over SNA LU6.2 node, or you can define a separate LU6.2 node to be used for Sockets over SNA. In this setup, we chose to define a separate local LU6.2 for Sockets over SNA use.

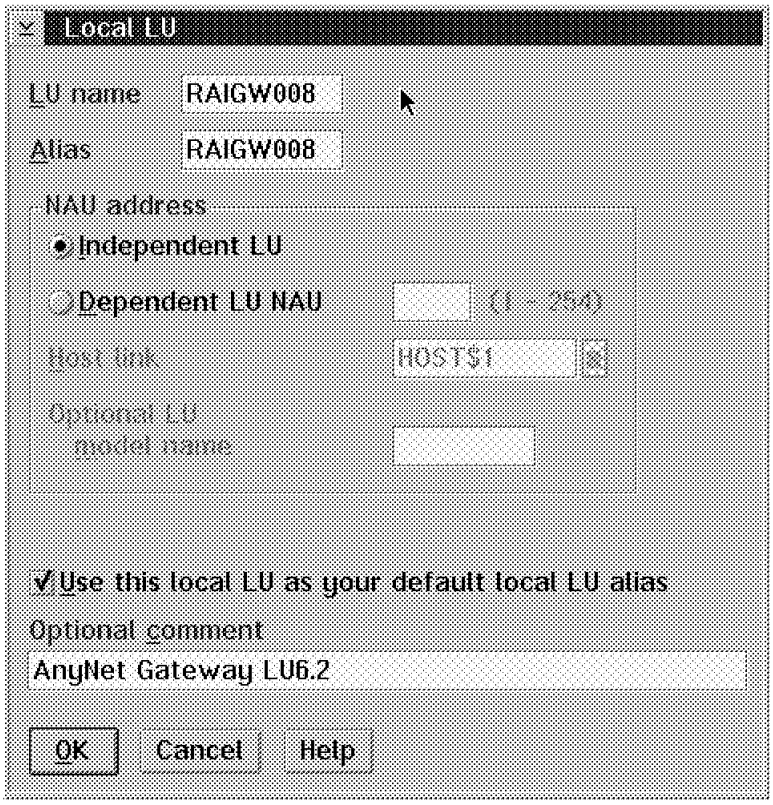


Figure 81. Local LU6.2 Node Definition

The LU name matches the LU name we specified in the AnyNet MVS initialization commands (see Figure 70 on page 107, istskmap for IP address 192.168.210.8).

To define AnyNet MVS as a partner LU, use the definitions in Figure 82.

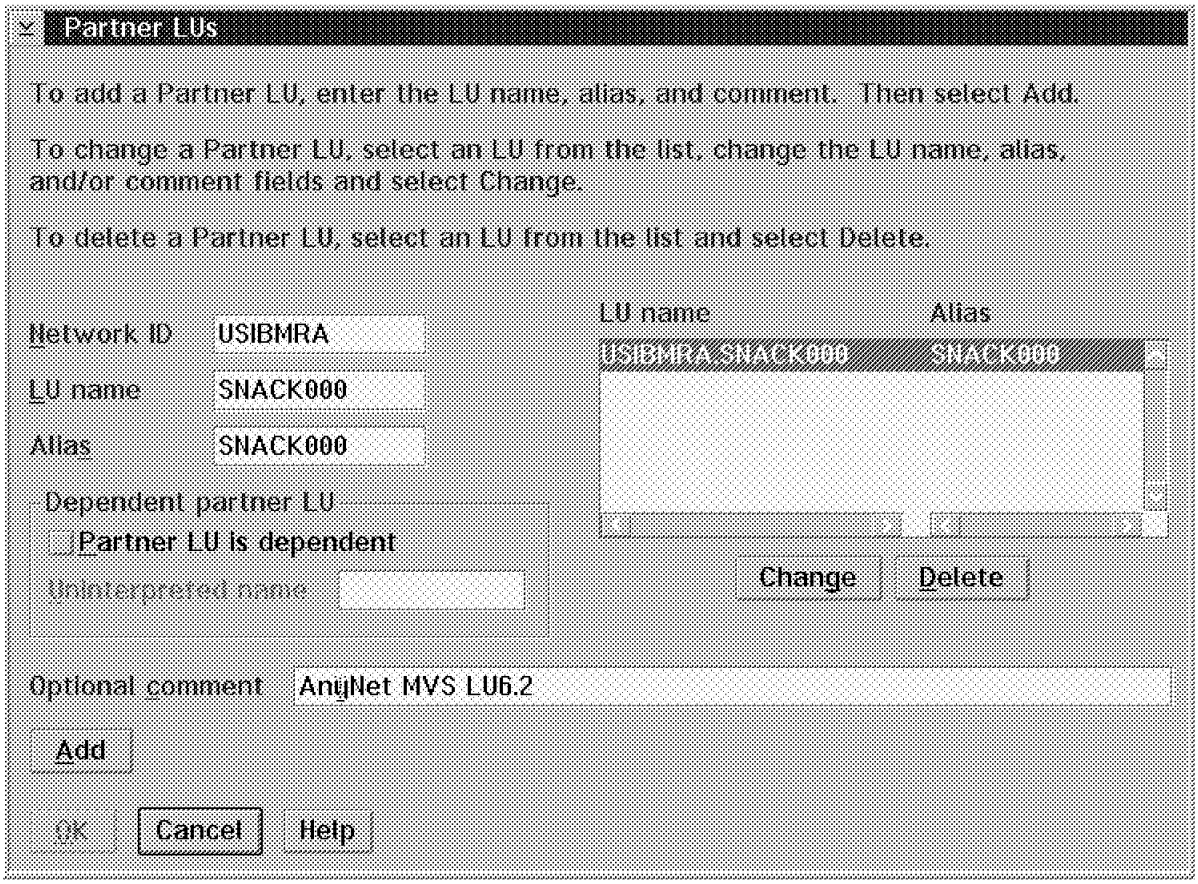


Figure 82. Partner LU Definition

At this point, all required SNA definitions are completed and you may return to the Communications Manager Configuration Definition panel.

To configure AnyNet/2 Sockets over SNA, select **Sockets** and **Configure** on the action bar. The following panels will initialize the IP - LU mapping tables and routing information. On the first panel, local node characteristics for the sna0 interface are defined. Enter the IP address you want to assign to AnyNet/2 Sockets over SNA and set the associated subnet mask.

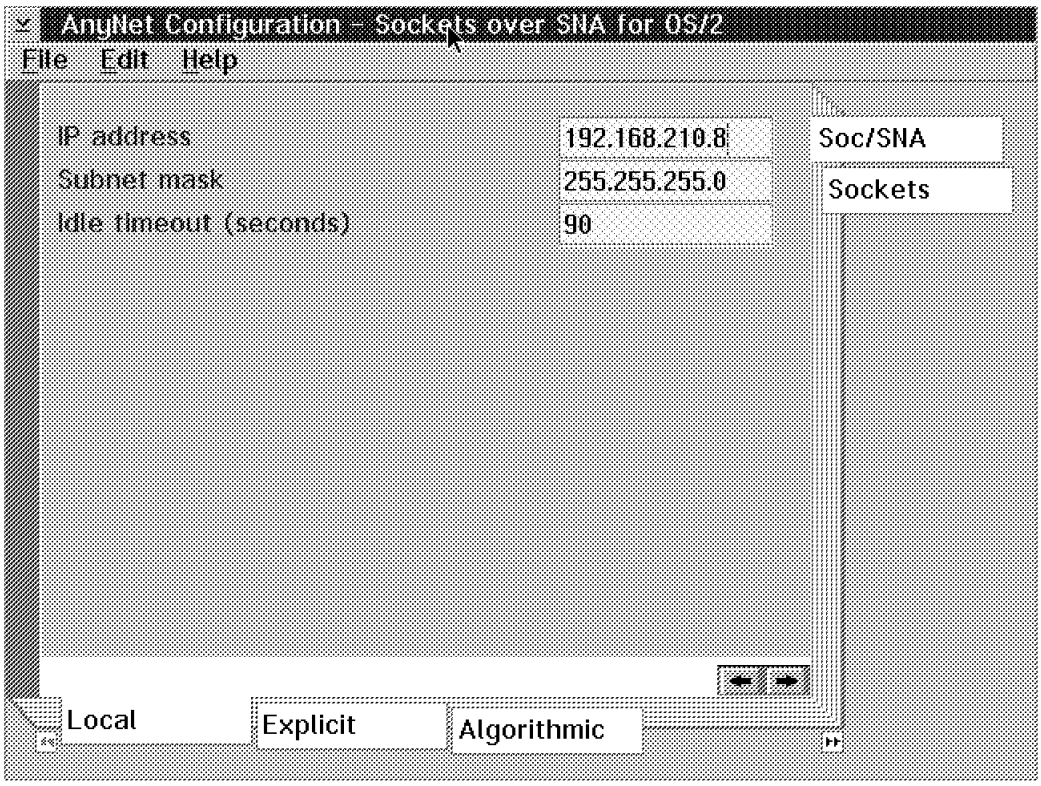


Figure 83. Define a Sockets over SNA sna0 Interface

In this setup, the local sna0 interface is assigned an IP address of 192.168.210.8, which again matches the AnyNet MVS initialization commands (see Figure 70 on page 107).

The subnet mask is really a net/subnet mask. If, as in our sample network, you use a separate class C network for the AnyNet network and you do not subnet this class C network, you still need to specify a subnet mask, which in this case corresponds to the class C net mask (255.255.255.0).

Choose between explicit and algorithmic IP-LU address mappings. In a small AnyNet network, explicit address mapping may be easier and clearer. For setting up a complex network using algorithmic address mappings, please refer to redbooks *IBM Communication Server for OS/2 Warp V4.90 Enhancements*, SG24-4587-00. and *AnyNet/2 Sockets over SNA*, GG24-4396-00. The following examples show how to define explicit address mappings.

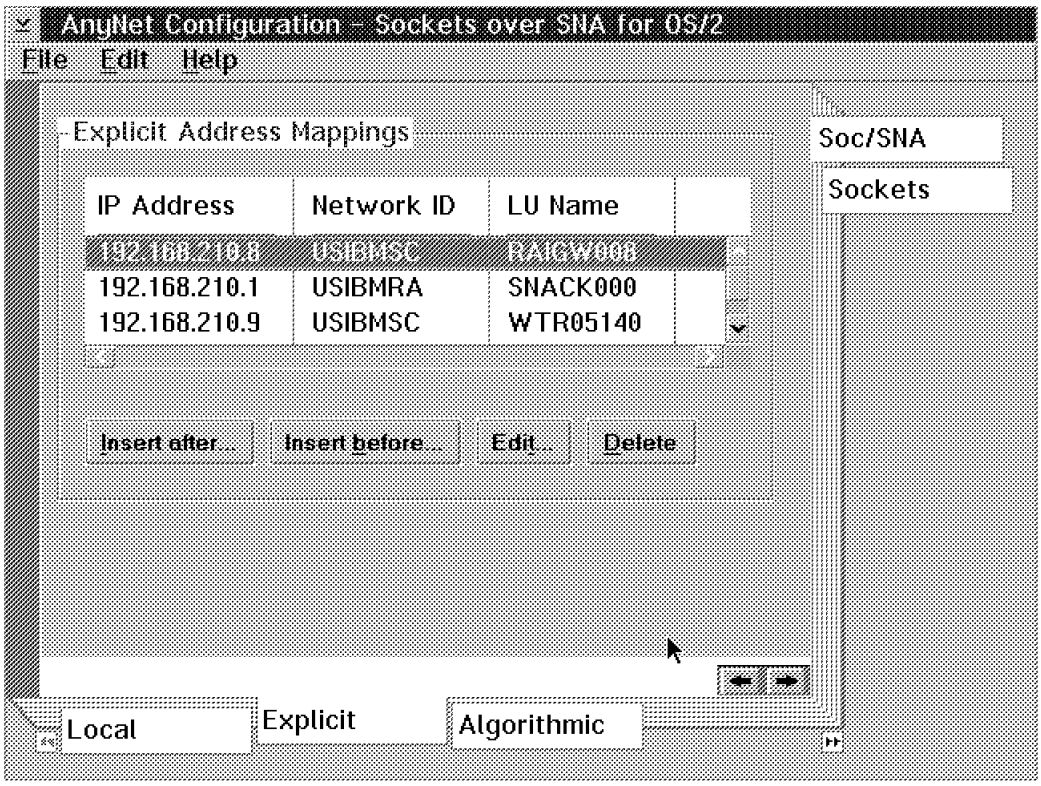


Figure 84. Define Explicit IP-LU-Address Mappings

The first entry is for the local LU6.2 name RAIGW008 that maps to the IP address we assigned to this AnyNet/2 node's sna0 interface (see Figure 83 on page 117).

The second entry defines our AnyNet MVS host. The values specified here must match the AnyNet MVS initialization commands (see Figure 70 on page 107, the istskifc command).

The last entry defines a second AnyNet/2 workstation in the AnyNet network.

Routing definitions are not required in our sample network. You would need routing definitions if you wanted to reach IP hosts that were accessible through other AnyNet gateway nodes in your AnyNet network.

Ensure that the default mode for Sockets over SNA is set to SNACKETS on the following panel.

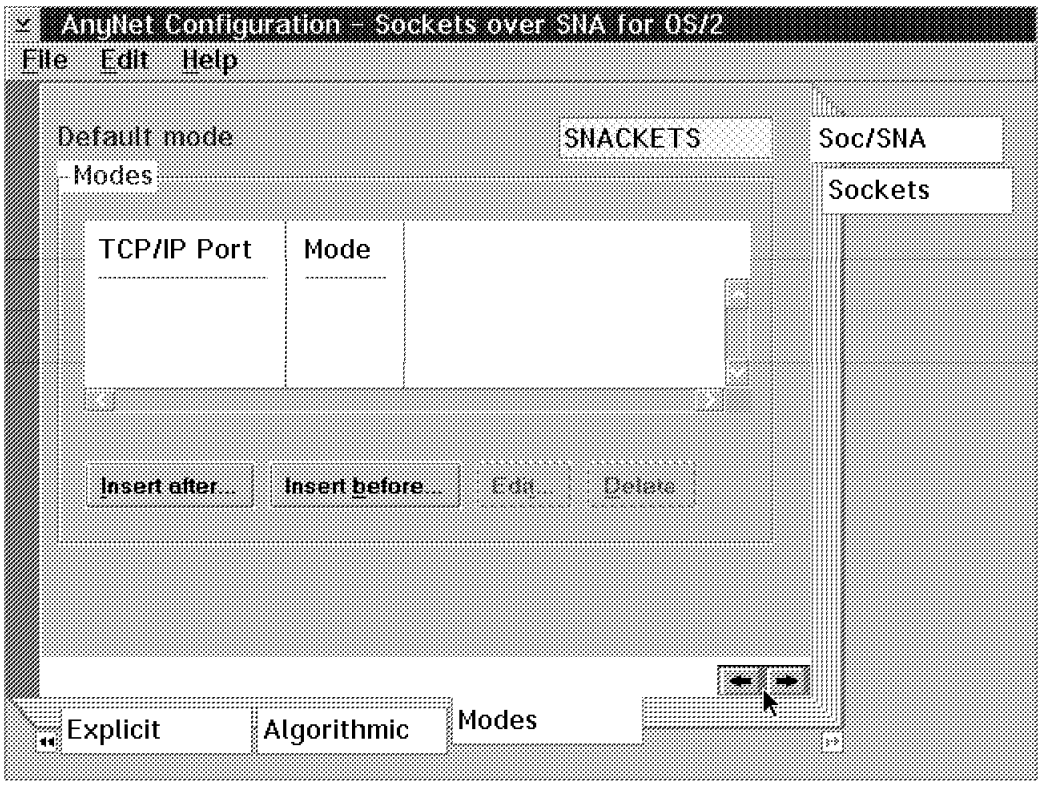


Figure 85. Select Sockets over SNA Default Mode

File and save all definitions for AnyNet/2 Sockets over SNA and verify your Communications Manager configuration. Any errors will be displayed in the FFST error log and have to be corrected before Communications Manager is restarted.

If your workstation has an IP interface in addition to its AnyNet interface, you can enable IP routing in the TCP/IP configuration parameters. By doing so, you turn your AnyNet/2 workstation into an AnyNet/2 gateway node.

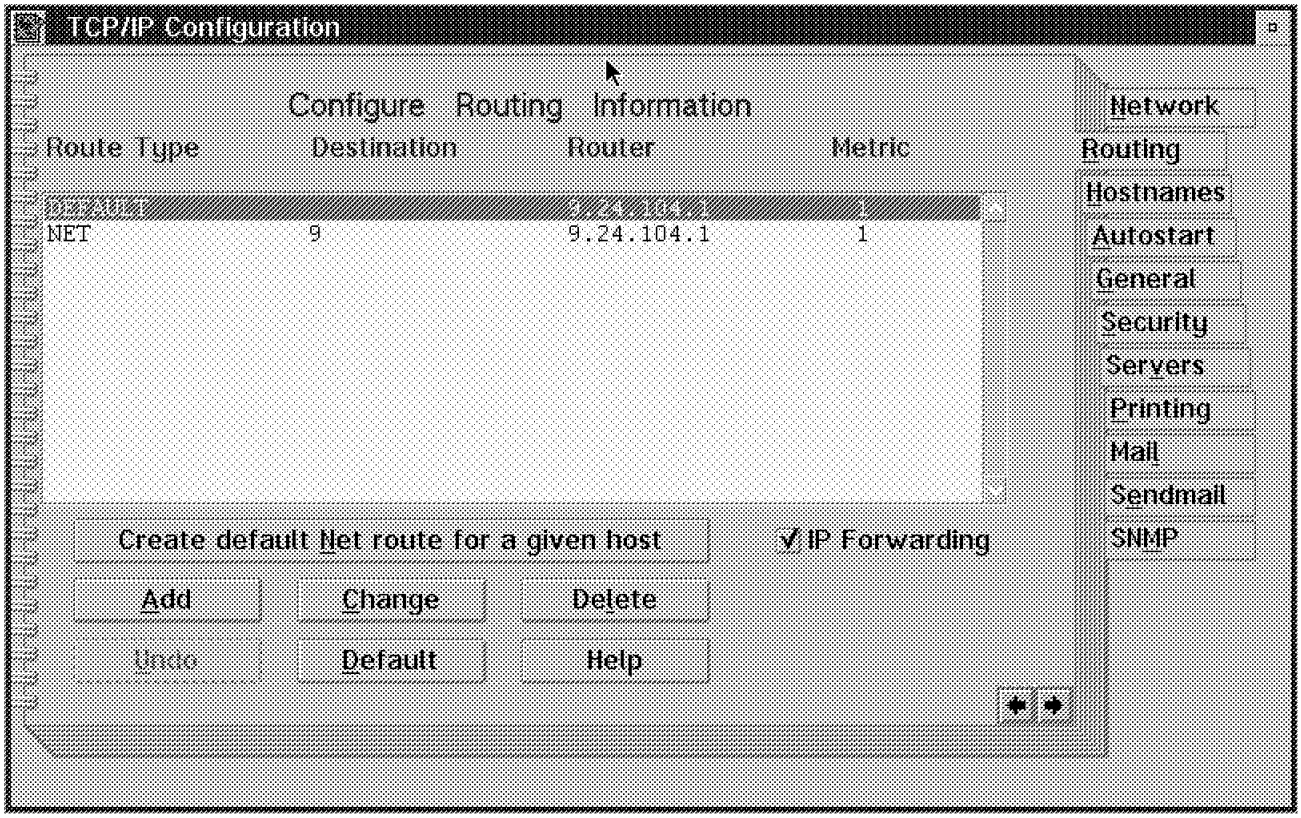


Figure 86. Activate IP Forwarding

AnyNet/2 will be started automatically if Communications Manager is restarted or the workstation rebooted. An active window running program SX.EXE on the desktop indicates that Sockets over SNA has been initialized and is running.

To check the generated entries in the IP-LU mapping table, issue the sxmap get command in an OS/2 window. To check that the sna0 interface is active, issue a netstat -a command.

```
[C:\]sxmap get  1
Address         Mask            SNA Network     LU Template
 -------------------------------------------------------------
192.168.210.9   FFFFFFFF        USIBMSC         WTRO5140
192.168.210.1   FFFFFFFF        USIBMRA         SNACKOOO
192.168.210.8   FFFFFFFF        USIBMSC         RAIGWOO8
-----------------------------------------------------------------
Address         Mask            IPX Network     IPX Node Address
 -----------------------------------------------------------------
No entries

[C:\]netstat -a
addr          127.0.0.1 interface 0 mask ff000000 broadcast           0.0.0.0
addr  2     9.24.104.79 interface 0 mask ffffff00 broadcast       9.24.104.255
addr  3   192.168.210.8 interface 4 mask ffffff00 broadcast           0.0.0.0
addr          127.0.0.2 interface 5 mask ff000000 broadcast           0.0.0.0
```

Figure 87. AnyNet/2 Configuration Verification

1 The sxmap get command displays the current IP to LU name mapping table.

2 This is the native IP network interface.

3 This is the Sockets over SNA sna0 interface.

If the sna0 interface is not active, start it with the following command:

```
ifconfig sna0 this-workstations-IP-address up.
```

At this point, a ping command can be issued to verify connections to a Sockets over SNA IP address.

A good tool to monitor the success of all your efforts is to start the Subsystem Management program in your Communications Manager Administration folder. Click on the **Details** pull-down, select **SNA Subsystem** and click on **LU6.2 sessions** to see the currently established LU6.2 sessions.

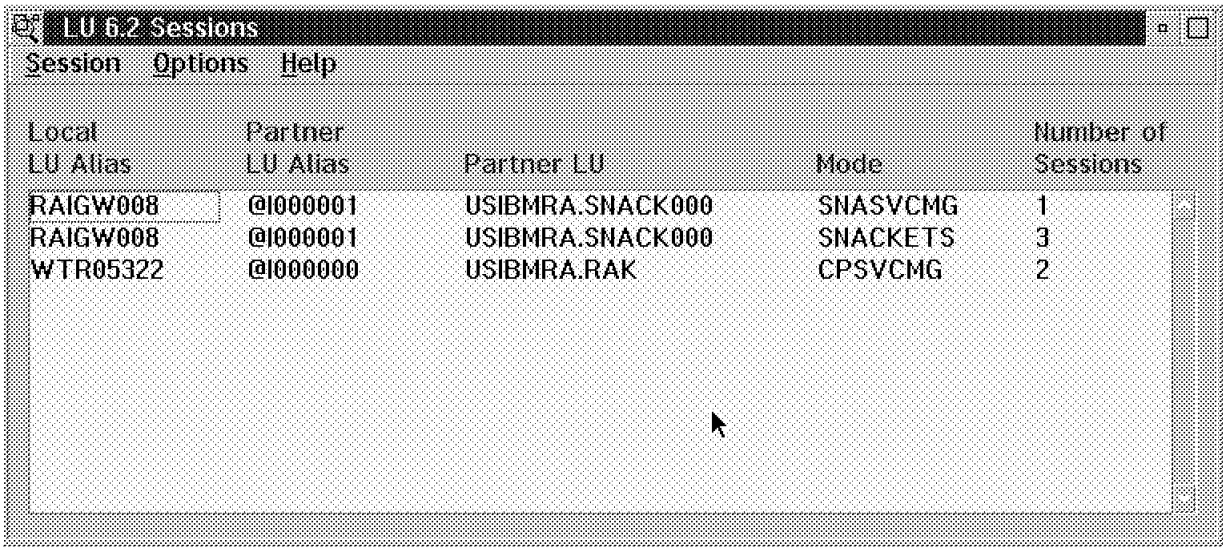


Figure 88. LU6.2 Session Status

*

An active control session to the AnyNet MVS host indicates that Sockets over SNA has been initialized and is running. The session that is using mode table entry CPSVCMG is the control point to control point session. The sessions between local LU RAIGW008 and partner LU SNACK000 is the sessions between your AnyNet/2 node and AnyNet MVS.

A new socket request, such as running a socket application or issuing a ping command, will create additional LU6.2 sessions. The Communications Manager SNA subsystem panel will be refreshed in a customizable interval and reflect any changes.

4.8 Start of OpenEdition, TCP/IP, AnyNet

When starting OpenEdition with a TCP/IP for MVS and an AnyNet MVS transport provider stack, you will find the following messages on SYSLOG and the JOBLOGs of the different started tasks:

4.8.1 OpenEdition Startup

When you start OpenEdition, for example with a START command from an MVS console, the following initialization messages will be displayed on SYSLOG:

```
S OMVS
$HASP100 OMVS     ON STCINRDR
IEF695I START OMVS     WITH JOBNAME OMVS     IS ASSIGNED TO USER
OMVSKERN, GROUP OMVSGRP
$HASP373 OMVS     STARTED
IEF403I OMVS - STARTED - TIME=12.56.00
BPXF013I FILE SYSTEM SMS.OMVS52.SA18.ROOT 338 1
WAS SUCCESSFULLY MOUNTED.
BPXF013I FILE SYSTEM SMS.IMW.SIMWHFS.SA18 339
WAS SUCCESSFULLY MOUNTED.
BPXF013I FILE SYSTEM SMS.TCPIP.XWIN.SA18 340
WAS SUCCESSFULLY MOUNTED.
BPXF013I FILE SYSTEM SMS.SIOE.LOCAL.SA18 341
WAS SUCCESSFULLY MOUNTED.
BPXF013I FILE SYSTEM SMS.SIOE.GLOBAL.SA18 342
WAS SUCCESSFULLY MOUNTED.
BPXF013I FILE SYSTEM SMS.DCEAS.SA18 343
WAS SUCCESSFULLY MOUNTED.
BPXF013I FILE SYSTEM SMS.EUV.LOCAL.SA18 344
WAS SUCCESSFULLY MOUNTED.
BPXF013I FILE SYSTEM SMS.EUV.GLOBAL.SA18 345
WAS SUCCESSFULLY MOUNTED.
BPXF013I FILE SYSTEM SMS.OMVS.USERS 346
WAS SUCCESSFULLY MOUNTED.
BPXF013I FILE SYSTEM SMS.OMVS.USER1 347
WAS SUCCESSFULLY MOUNTED.
BPXF203I DOMAIN AF_UNIX WAS SUCCESSFULLY ACTIVATED. 2
BPXF203I DOMAIN AF_INET WAS SUCCESSFULLY ACTIVATED. 2
BPXI004I OMVS INITIALIZATION COMPLETE 3
```

Figure 89. OpenEdition Startup

1 The message BPXF013I is issued for each HFS that has been successfully mounted.

2 The message BPXF203I indicates the specified addressing family can now be used.

3 If the initialization of OpenEdition has been successfully completed, the message BPXI004I is displayed.

4.8.2 TCP/IP for MVS Startup

When a TCP/IP for MVS transport provider stack is started, the following messages will be displayed on SYSLOG and the JOBLOG of the appropriate TCP/IP stack:

```
S T18OTCP

IEF403I T18OTCP - STARTED - TIME=11.01.28
EZY1876I TCPIP started with parameter TCPIP,ERRFILE(SYSERR),HEAP(512),
NOSPIE/.
EZB7455I MSGOPEN: OPEN OF SYSPRINT SUCCESSFUL, USING PRIMARY DATASET
EZB7455I MSGOPEN: OPEN OF SYSDEBUG SUCCESSFUL, USING PRIMARY DATASET
EZB6473I TCP/IP initialization complete. 1
START T18OROUT  2
START T18ODNS   2
EZY2140I OpenEdition-TCP/IP connection established for T18OTCP 3
BPXF206I ROUTING INFORMATION FOR TRANSPORT DRIVER T18OTCP HAS BEEN
INITIALIZED OR UPDATED. 4
```

Figure 90. TCP/IP for MVS Startup Messages

1 The message EZB6473I indicates the successful initialization of a TCP/IP for MVS stack.

2 The RouteD started task and the name server started task are started automatically via the AUTOLOG section in the PROFILE data set.

3 The message EZY2140I specifies the started task user ID of the transport provider stack, for which a connection to OpenEdition has been established.

4 The message BPXF206I indicates that the converged sockets pre-router has successfully obtained a copy of the IP layer routing table from the transport provider stack. This message will be re-issued every time the IP layer routing tables are updated by the RouteD server.

4.8.3 AnyNet MVS Startup

When the AnyNet MVS transport provider stack is started and initialized using the initialization commands, the following messages will be displayed on SYSLOG:

```
S RAISOCK
IEF403I RAISOCK - STARTED - TIME=13.15.11
ISU1501I SOCKETS-OVER-SNA RAISOCK INITIALIZATION COMPLETE FOR V4R3 1

S RAISOCKI
IEF403I RAISOCKI - STARTED - TIME=13.15.24
BPXF206I ROUTING INFORMATION FOR TRANSPORT DRIVER RAISOCK HAS BEEN
INITIALIZED OR UPDATED. 2
IEF404I RAISOCKI - ENDED - TIME=13.15.33
```

Figure 91. AnyNet MVS Startup Messages

1 The message ISU1501I indicates the successful initialization of the AnyNet MVS stack.

2 The message BPXF206I indicates here that the OpenEdition routing information for the specified transport provider stack has been initialized.

4.8.4 Startup Completion

After OpenEdition and all transport provider stacks (in this case one TCP/IP stack, and one AnyNet stack) are successfully started and all listener processes have been started, a DISPLAY command for the OMVS address space will show the following result:

```
D OMVS,A=ALL
BPXO001I 10.13.03 DISPLAY OMVS 411
OMVS     ACTIVE                                          BPXPRM18
USER     JOBNAME  ASID        PID       PPID STATE   START     CT_SECS
OMVSKERN OMVSINIT 0036          1          0 MRI   18.02.45       .097
  SERVER=Init Process                      AF=    0 MF=65535 TYPE=FILE
T18OTCP  T18OTCP  0046      65539          1 MR    18.04.48   1291.802
RAISOCK  RAISOCK  0052     327685          1 MR    18.04.08    595.028
  SERVER=SNACKETS                          AF=    0 MF=65535 TYPE=FILE
OMVSKERN SYSLOGD4 002C     262150          1 1FI   18.03.35     23.071
  LATCHWAITPID=         0 CMD=/usr/sbin/syslogd
OMVSKERN INETD5   0037          7          1 1FI   18.03.37      1.951
  LATCHWAITPID=         0 CMD=/usr/sbin/inetd /etc/inetd.conf
```

Figure 92. OpenEdition SYSLOG Display

4.9 Stopping OMVS, TCP/IP and AnyNet

When AnyNet MVS Sockets over SNA is terminated, the following messages will be displayed on SYSLOG:

```
P RAISOCK

ISU1511I SOCKETS-OVER-SNA RAISOCK SHUTDOWN INITIATED 1
BPXF207I ROUTING INFORMATION HAS BEEN DELETED FOR TRANSPORT DRIVER
RAISOCK. 2
ISU1538I SOCKETS-OVER-SNA MESSAGE GROUP: RAISOCK 516
ISU1548I SOCKETS-OVER-SNA IS CLOSING
ISU1516I END OF SOCKETS-OVER-SNA MESSAGE GROUP
ISU1513I SOCKETS-OVER-SNA RAISOCK SHUTDOWN COMPLETE 3
IEF404I RAISOCK - ENDED - TIME=13.25.51
```

Figure 93. AnyNet MVS Termination Messages

1 The message ISU1511I indicates Sockets over SNA that a STOP command has been issued and Sockets over SNA is beginning to terminate.

2 The message BPXF206I indicates here that the OpenEdition routing information for the specified transport provider stack has been deleted. The transport provider stack is no longer active and the connection to OpenEdition has been cleared.

3 The message ISU1513I is issued when AnyNet MVS Sockets over SNA is terminated.

```
P T18OTCP

IEF404I T18OROUT - ENDED - TIME=13.35.54
IEF404I T18ODNS - ENDED - TIME=13.35.54
EZB4465E PCCA3 shutting down:
EZB5047E IUCV shutting down:
BPXF207I ROUTING INFORMATION HAS BEEN DELETED FOR TRANSPORT DRIVER
T18OTCP. 1
EZY1877I TCPIP shutdown is complete, rc = 0. 2
IEF404I T18OTCP - ENDED - TIME=13.36.26
```

Figure 94. TCP/IP for MVS Termination Messages

1 The message BPXF206I indicates that the OpenEdition routing information for the specified transport provider stack has been deleted. The transport provider stack is no longer active, and the connection to OpenEdition has been cleared.

2 The message EZY1877I indicates that TCP/IP for MVS has terminated. If no errors occurred during shutdown, then rc = 0.

```
P OMVS

BPXI016I OMVS IS BEGINNING TO TERMINATE
IEA989I SLIP TRAP ID=X422 MATCHED.  JOBNAME=INETD5  , ASID=002E. 1
IEA989I SLIP TRAP ID=X422 MATCHED.  JOBNAME=FTPD1   , ASID=0021.
IEA989I SLIP TRAP ID=X422 MATCHED.  JOBNAME=OMVSINIT, ASID=0023.
IEA989I SLIP TRAP ID=X422 MATCHED.  JOBNAME=SYSLOGD4, ASID=002A.
IEA989I SLIP TRAP ID=X422 MATCHED.  JOBNAME=CS2201SR, ASID=002D.
IEF404I OMVS - ENDED - TIME=13.43.33
```

Figure 95. OpenEdition Termination Messages

1 Currently running processes at the time of OMVS shutdown are abended with a 422 abend, as indicated in the IEA989I messages.

4.10 Recycling Transport Providers

You need to start the default transport provider before you try to start any of your other transport providers. If you, for example, have defined a TCP/IP for MVS stack as your default transport provider, but start AnyNet MVS before your TCP/IP for MVS stack has been started, AnyNet MVS will not be able to establish a transport provider connection with the converged sockets physical file system.

```
BPXF205I UNABLE TO ESTABLISH A CONNECTION TO TRANSPORT DRIVER RAISOCK
FOR ROUTING INFORMATION.  RETURN CODE = 00000070, REASON CODE =
12FC0296. 1
```

Figure 96. Starting Transport Providers in Wrong Sequence

The return code of X'70' means that a resource is temporarily unavailable. The last four digits of the reason code X'0296' means that TCP/IP is not active, which could be misleading; the real reason for this error message is that we have not yet started the default transport provider.

If you have two transport providers connected to OMVS, of which one is the default, and you stop the default, you can no longer start new socket programs in OpenEdition. If you try to do so, the socket programs will get the same error codes back as mentioned above, when they try to open a new socket.

To summarize:

1. Always start your default transport provider first.
2. If you have to stop and restart your default transport provider, be aware that new socket connections can not be established while the default transport provider is disconnected from OMVS.
3. You may stop and restart a non-default transport provider without any impact on the other transport providers.
4. After you have started a transport provider (be it the default or a non-default) you have to recycle your server listener processes (InetD, FTPD, Web servers, and your homegrown socket server programs).

4.11 Automation of OMVS Operations

In the sample setup that was established during the creation of this book, we implemented some basic NetView for MVS/ESA message automation to assist us with the following tasks:

- Start of the OMVS environment

  1. We manually start OMVS via an S OMVS operator command. In a stable production environment, this start command could be initiated either via your SYS1.PARMLIB command member (COMMNDxx) or via NetView initiated commands.
  2. When message BPXI004I occurs, the default transport provider is started via an MVS start command: S T18OTCP.

```
*
* BPXI004I OMVS Initialization has completed
* Start the default transport provider: T18OTCP
*
IF MSGID = 'BPXI004I' THEN BEGIN;
     IF DOMAINID = 'RAIAO'
     THEN EXEC( CMD('MVS S T18OTCP') ROUTE(ONE SYSAUTO))
          DISPLAY(Y) NETLOG(Y) SYSLOG(Y);
END;
```

  3. When the EZY2140I message occurs, we start a NetView REXX program called OERECYCL, which controls the stop and start of all our OpenEdition listener processes.

```
*
* TCPIP Connection with OE established, recycle listener processes
* Must only match for the OE stack: T18OTCP
*
* EZY2140I OpenEdition-TCP/IP connection established for T18OTCP
* T: 1              2             3            4      5     6
*
IF MSGID = 'EZY2140I' & TOKEN(6) = 'T18OTCP' THEN BEGIN;
     IF DOMAINID = 'RAIAO'
     THEN EXEC( CMD('OERECYCL') ROUTE(ONE SYSAUTO))
          DISPLAY(Y) NETLOG(Y) SYSLOG(Y);
END;
```

- Start of the AnyNet MVS transport provider

  1. We start AnyNet MVS manually via an S RAISOCK operator command. In our setup, the AnyNet MVS stack was not up all the time. We started and stopped it when required for our testing.
  2. When message ISU1501I occurs, the AnyNet MVS initialization job (RAISOCKI) is started and the same NetView REXX as mentioned earlier, the OERECYCL REXX program, is started.

```
*
* AnyNet MVS is up. Start AnyNet initialization job and
* recycle the OE Listener processes.
*
IF MSGID = 'ISU1501I' THEN BEGIN;
     IF DOMAINID = 'RAIAO'
     THEN EXEC( CMD('MVS S RAISOCKI') ROUTE(ONE SYSAUTO))
          EXEC( CMD('OERECYCL') ROUTE(ONE SYSAUTO))
        DISPLAY(Y) NETLOG(Y) SYSLOG(Y);
END;
```

See Figure 97 on page 128 for an overview of the automation components that we implemented. All the components are included in Appendix B, “Operations Automation Code Samples” on page 205.

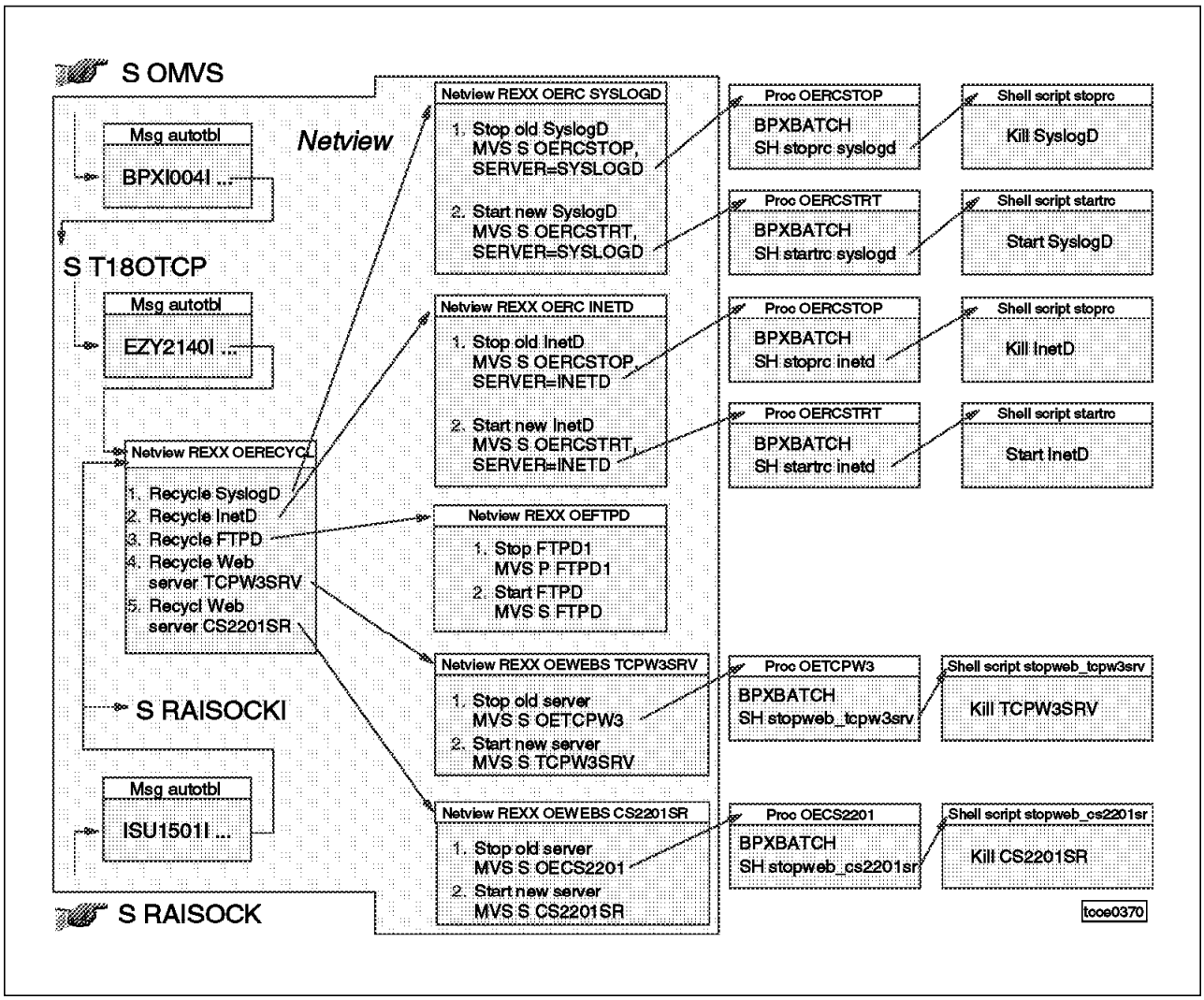


Figure 97. Automation Overview

It is not currently possible to use OpenEdition services in a REXX program that executes in NetView for MVS/ESA. We wanted to keep as much of the automation logic as possible within the NetView address space, but in order to handle the situations where OpenEdition shell commands were needed, we started JCL procedures with the BPXBATCH utility to run a few shell scripts.

Chapter 5. TCP/IP for MVS OpenEdition Applications Feature

The TCP/IP for MVS OpenEdition Applications Feature provides a set of programming interfaces and standard TCP/IP applications to enhance interoperability between an OpenEdition environment and clients or servers in a TCP/IP network. It relies exclusively on OpenEdition (MVS/ESA SP 5.2.2 or OS/390 with OpenEdition is a prerequisite for installation and use). The TCP/IP for MVS OpenEdition Applications Feature is available as a no-charge feature of TCP/IP Version 3 Release 1 for MVS. The new servers that are supplied with the TCP/IP for MVS OpenEdition Applications Feature can be used concurrently with the already available TCP/IP for MVS servers for the traditional MVS environment.

This chapter includes the following topics:

- 5.1, OpenEdition Applications from TCP/IP for MVS
- 5.2, SyslogD
- 5.3, InetD Customization
- 5.4, TelnetD Customization
- 5.5, REXECD and RSHD Customization
- 5.6, REXEC Client
- 5.7, FTPD Customization
- 5.8, ONC/RPC Programming
- 5.9, X-Windows Programming

© Copyright IBM Corp. 1996

5.1 OpenEdition Applications from TCP/IP for MVS

The following new functions are supplied with the TCP/IP for MVS OpenEdition Applications Feature:

- OE SyslogD server

  Supplies logging functions for programs that execute in the OpenEdition environment. SyslogD is used by more of the servers from the TCP/IP for MVS OpenEdition Applications Feature, but can be used by any program in OpenEdition.

- OE TelnetD server

  Allows hosts in the IP network to log on to the OpenEdition shell environment directly, without going through TSO. The OE TelnetD server supports both line-mode and raw-mode telnet connections.

- OE FTPD server

  The OE FTPD server allows remote users to transfer files directly in and out of the Hierarchical File System in OpenEdition. The server, in addition to support for Hierarchical File System files, also supports standard MVS data sets. This FTPD server includes all the functions of the non-OE C-FTP server in TCP/IP Version 3 Release 1 for MVS, including SQL query support and JES file type support.

- OE REXEC client

  Allows OpenEdition shell users to execute commands on a remote system.

- OE REXECD server

  Provides server functions in the OpenEdition environment for remote commands based on the remote execution (REXEC) protocol. Commands are executed as OpenEdition shell commands.

- OE RSHD server

  Provides server functions in the OE environment for remote shell clients based on the remote shell (RSH) protocol. Commands are executed as OpenEdition shell commands.

- ONC/RPC programming libraries

  Supplies an application programming interface (API) used for remote procedure call server and client applications in OE and includes a library of procedures and external data representation.

- OSF/Motif and X-Windows programming libraries

  Provides X-Windows X11R6 and OSF/Motif 1.2.4 programming interfaces to be used by X-Windows client programs in the OpenEdition environment.

The TCP/IP for MVS OpenEdition Applications Feature is supplied as a no-charge feature of TCP/IP Version 3 Release 1 for MVS. The basic SMP/E installation will only be possible if TCP/IP Version 3 Release 1 for MVS is installed in the SMP/E environment. The use of the TCP/IP for MVS OpenEdition Applications Feature, however, is not dependent on TCP/IP for MVS. Both TCP/IP for MVS and AnyNet MVS as supported transport providers for OpenEdition can be used to access the OE applications in an MVS/ESA SP 5.2.2 environment.

The individual parts of the TCP/IP for MVS OpenEdition Applications Feature are installed into both the Hierarchical File System and TCP/IP for MVS product

libraries. See Table 9 on page 131 for an overview of where the individual parts are installed.

Table 9. TCP/IP for MVS OpenEdition Applications Feature Parts as They are Supplied With TCP/IP Version 3 Release 1 for MVS

Part	Hierarchical File System Location	MVS Library Location
SyslogD server	/usr/sbin/syslogd•	SYS1.LINKLIB(SYSLOGD)
TelnetD server	/usr/sbin/otelnetd•	SEZALINK(OTELNETD)
REXECD server	/usr/sbin/rexecd•	SEZALINK(REXECD)
RSHD server	/usr/sbin/rshd•	SEZALINK(RSHD)
REXEC client	/bin/orexec•	SEZALINK(OREXEC)
FTPD server • Listener • Server	 • /usr/sbin/ftpd• • /usr/sbin/ftpdsrv•	 • SEZALINK(FTPD) • SEZALINK(FTPDSRV)
ONC/RPC • Port mapper • RPCGEN • RPCINFO • Header files • Library archive • Sample code	 • /bin/oportmap• • /bin/orpcgen• • /bin/orpcinfo• • /usr/include/rpc • /usr/lib/librpclib.a • /usr/lpp/tcpip/rpc/samples	 • SEZALINK(OPORTMAP) • SEZALINK(ORPCGEN) • SEZALINK(ORPCINFO) • n/a • n/a • n/a
X-Windows • Header files • Library archives • Uil compiler • Sample code	 • /usr/include • /usr/lib • /bin/X11/uil • /usr/lpp/tcpip/X11R6/Xamples	n/a

Note:

• These executable files all have the sticky bit set in the Hierarchical File System, which means that their MVS library counterparts are being executed.

Please note that modules are being installed into your existing TCP/IP for MVS SEZALINK library during installation of the TCP/IP for MVS OpenEdition Applications Feature. The TCP/IP for MVS programming directory instructs you to add SEZALINK to your system link list. Some installations have also added the SEZATCP library to their system link list, which removes the requirement for STEPLIB DD statements in the server JCL procedures. If you are running with both SEZALINK and SEZATCP on your system link list, you must be aware that one module name is used in both libraries, but for different programs; it is the RSHD module name:

- In the SEZATCP library, this is the combined non-OE REXECD and RSHD server load module.
- In the SEZALINK library, this is the OpenEdition RSHD server load module. In this library, the RSHD name is an alias for load module EZARSDCS.

There are different ways you can resolve this; the easiest is probably the following:

1. Make sure your SEZALINK library is specified before your SEZATCP library in your LNKLSTxx SYS1.PARMLIB member.

2. Add a STEPLIB DD statement to your non-OE remote execution server JCL procedure, so this procedure executes the SEZATCP version of the RSHD program.

Note: Both the names and the location of the executable files may change in future releases of the TCP/IP for MVS OpenEdition Applications Feature.

5.2 SyslogD

SyslogD is a server process that has to be started as one of the first processes in your OpenEdition environment. Other servers, including InetD use SyslogD for logging purposes during initialization. Servers on the local system use AF_UNIX sockets for communication with SyslogD. If SyslogD is not started, when an application tries to send log data to SyslogD, the log data will appear on the MVS console. See Figure 98 for an overview of how SyslogD operates in the OpenEdition environment.

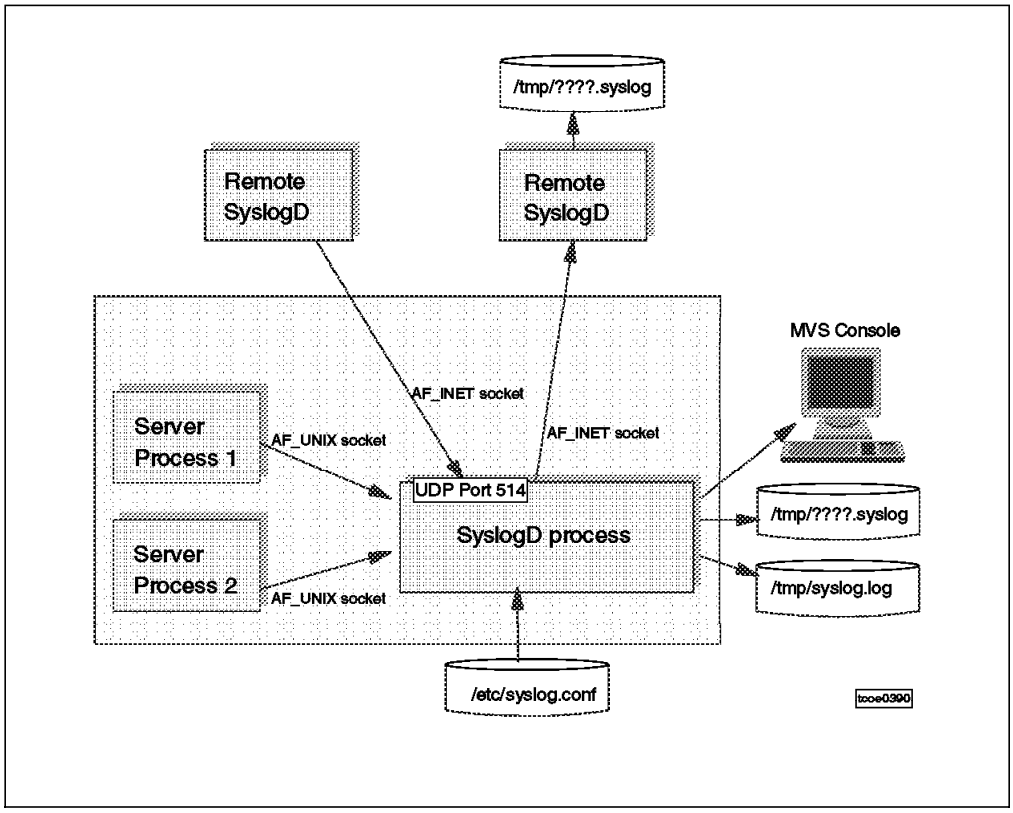


Figure 98. SyslogD Overview

SyslogD is able to receive log data from other SyslogD servers in your network. For that purpose SyslogD creates a UDP socket and binds it to port 514 during initialization. This has a couple of consequences:

1. UDP port 514 must be reserved for OMVS in your TCP/IP for MVS PROFILE data set for your AF_INET transport provider stack.

```
PORT
   . . .
   514 UDP OMVS                 ; OE SyslogD Server
   . . .
```

2. If there is no AF_INET transport provider connected to OE when SyslogD is started, SyslogD′s bind() socket call fails. SyslogD does not abort its initialization for this reason, but it does not retry the bind() call either, which means that SyslogD in this situation will not be able to receive log data from other SyslogD servers in your network. To cope with this situation, we decided to treat SyslogD the same way we treated all other AF_INET socket server programs. We did not start SyslogD until after our default AF_INET transport provider had connected, and we re-cycled SyslogD along with other socket servers, when one of our AF_INET transport provider stacks re-connected to OpenEdition after a stack restart.

The actual start of SyslogD is done via a small shell script that allows us to set jobname and eventually also pass proper environment variables to the SyslogD process:

```
#
# Start the SYSLOGD daemon
#

export _BPX_JOBNAME='SYSLOGD'
export _CEE_RUNOPTS='ALL31(ON)'
/usr/sbin/syslogd -f /etc/syslog.conf &

echo -- /u/alfredc/restart_syslogd.sh script executed, date
```

In our implementation this shell script is executed by a BPXBATCH job that is started from NetView for MVS/ESA.

We could have used a JCL procedure with the name SYSLOGD to start the SyslogD server as a POSIX(ON) program, but by starting it via a shell script we are able to see the program name on a ps -ef shell command. We use this to find the process ID of the current SyslogD process, when we need to stop SyslogD in order to restart it after a transport provider has reconnected.

```
/u/alfredc: >ps -efc
     UID        PID       PPID  C    STIME TTY       TIME CMD
       0          1          0  -   May 24 ?         0:00
 T18OTCP      65538          1  -   May 24 ?         7:58
       0    2686980    1507339  - 10:57:57 ttyp0000  0:00 /bin/ps
       0     524293          1  -   May 24 ?         0:05
       0    1114118         18  - 10:54:19 ?         0:06 /usr/sbin/otelnetd
       0     393223          1  -   May 24 ?         0:06
       0     131081          1  -   May 24 ?         7:10
       0    1507339    1114118  - 10:54:45 ttyp0000  0:01 /bin/sh
       0    1114125          1  - 10:49:44 ?         0:17
       0      65550          1  -   May 24 ?         0:00
       0         17          1  -   May 24 ?         1:08 /usr/sbin/syslogd
       0         18          1  -   May 24 ?         0:00 /usr/sbin/inetd
/u/alfredc: >
```

Please refer to 4.11, “Automation of OMVS Operations” on page 126 for details on the operational aspects of starting and stopping SyslogD.

As other servers that bind sockets to reserved port numbers, SyslogD runs under a user ID with a UID=0 (in our setup, OMVSKERN).

The processing of SyslogD is controlled via a configuration file called /etc/syslog.conf. This file can be used to define specific logging conditions and output destinations for error messages, authorization violation messages and trace data for systemwide use.

All log files used by SyslogD must be created in the Hierarchical File System before SyslogD is started.

The SyslogD configuration file allows you to set up logging rules depending on:

- A logging *facility* name
- A logging *priority* code

Both the facility name and the priority code are passed on the logging request from an application when it wants to log a message.

The following facility names and priority codes are predefined in the SyslogD implementation.

The facility names are:

kern	Messages generated by the OE Kernel address space.
user	Messages from anyone that does not fall within any of the other categories (the default facility name).
mail	Messages from the mail system.
news	Messages from the Usenet network news system.
uucp	Messages from the UUCP system.
daemon	This facility name is generally used by server processes. Both the FTPD server, the RSHD, and the REXECD server use this facility name to log server-related messages.
auth	Messages about authorization.
authpriv	Same as auth.
cron	Messages from the CRON daemon or the AT command.
lpr	Messages from the line printer system.
local0-7	These name are meant for local use. The TelnetD server in the TCP/IP for MVS OpenEdition Applications Feature uses facility name local1 for its log messages.
*****	Placeholder used to represent all facility names.

The priority codes are:

emerg	Emergency messages. System is becoming unusable.
panic	Same as emerg.
alert	Immediate action is required.
crit	Critical condition. A device or a component is becoming unusable.
error	Error condition.
warning	Warning condition.
notice	Normal, but significant condition.
info	Information.
debug	Debugging messages.
none	Placeholder used to represent none of the priorities.
*****	Placeholder used to represent all priorities.

The lowest priority is debug. The highest is emerg.

You combine facility name and priority codes in the entries in the /etc/syslog.conf file and for each combination you specify a destination for log messages that belong to the specified combination:

```
facility_name.priority_code  destination
```

The destination can be any of the following:

- A file in the Hierarchical File System

  ```
  facility_name.priority_code  /tmp/syslogd/auth.log
  ```

- One or more local users

  ```
  facility_name.priority_code  user1,user2
  ```

  You may send the messages to all logged-in users by specifying an asterisk as user name:

  ```
  facility_name.priority_code  *
  ```

- A SyslogD server on another host

  ```
  facility_name.priority_code  @myaixserver
  ```

- The MVS console

  ```
  facility_name.priority_code  /dev/console
  ```

Priority code specifications include all higher priorities. If you specify a priority code of, for example crit, messages of this priority plus messages with alert, panic and emerg priorities will be logged at the specified destination. To send all messages with a priority of crit or higher to a user ID of OPER1, specify the following rule in /etc/syslog.conf:

```
*.crit  OPER1
```

A message may be logged in more destinations, depending on your rules in /etc/syslog.conf. To capture all messages from facility name daemon into one file and all messages with a priority of crit or higher into another file, specify the following:

```
daemon.*  /tmp/syslogd/daemon.log
*.crit    /tmp/syslogd/crit.log
```

If a server sends a message to SyslogD with a facility name of debug and a priority code of alert, the above rules will log the message into both the daemon.log and the crit.log file.

One of the priority codes has a special meaning; it is the *none* priority code. If you include this priority code in a rule, it means: do *not* select any messages. What is the purpose of such a rule? If you want to log all messages from facility name local1 into one file and all from daemon into another and then everything else into a third, you can use the following rule set:

```
local1.*                      /tmp/syslogd/local1.log
daemon.*                      /tmp/syslogd/daemon.log
*.*;local1.none;daemon.none   /tmp/syslogd/the_rest.log
```

It is not possible to define logging conditions related to a jobname, process name or pid. All messages that belong to the same facility or priority class will be logged in the same SyslogD logging file, no matter what server task has issued the message.

```
#
# facility-name.priority  destination
# ----------------------  ----------------------------------------
#
# All alert messages (and above
# priority messages) go to the
# MVS console
#
*.alert                   /dev/console
#
#
# All authorization messages
# go to auth.log
#
auth.*                    /tmp/syslogd/auth.log
#
# All error messages (and
# above priority messages) go
# to error.log
#
*.err                     /tmp/syslogd/error.log
#
# All debug messages (and
# above priority messages) from
# telnet go to telnet.debug
#
local1.debug              /tmp/syslogd/telnet.debug
#
# All ftpd, rexecd, rshd
# debug messages (and above
# priority messages) go
# to server.debug
#
daemon.debug              /tmp/syslogd/server.debug
#
# Everything not directed to
# a destination above, is directed
# to garbagecan.log (so we don't loose
# anything important)
#
*.*;local1.none;daemon.none;auth.none /tmp/syslogd/garbagecan.log
```

Figure 99. Sample /etc/syslog.conf File

Your own OpenEdition application programs can use the logging facilities of the SyslogD server. Your C program must include the syslog.h header file. The program can then use the following functions to open a log facility, send log messages to SyslogD and close the facility again:

```
#include <syslog.h>

openlog("oec", LOG_PID, LOG_LOCAL0); 1
syslog(LOG_INFO, "Hello from oec");  2
closelog(); 3
```

1 Open a log facility name of local0. Prefix each line in the log file with the program name (oec) and the process ID.

2 Log an info priority message of the specified content.

3 Close the log facility name again.

The above statements resulted in the following line in the log file:

```
May 26 11:27:51 mvs18oe oec[3014660]: Hello from oec
```

For more information on the syslog function, please see *Stevens: Advanced Programming in the UNIX Environment*, published by Addison-Wesley.

A kill pid -SIGHUP command can be used to force SyslogD to re-read its configuration file and activate any modified parameters without stopping SyslogD.

SyslogD just keeps appending log messages to the files you have specified in your /etc/syslog.conf. You need some technique to periodically delete unwanted messages or offload the current log files to another location. If you update the log files from an archive process, you need to stop SyslogD while you do so. To avoid stopping SyslogD during archive and cleanup, you can create two SyslogD configuration files: one called /etc/syslog.conf.a and another called /etc/syslog.conf.b. The two files are equal except that all log files end with either an a or b.

If your current /etc/syslog.conf file was created from your syslog.conf.a file, then you can copy your syslog.conf.b file to /etc/syslog.conf and send SyslogD a SIGHUP signal, which will make SyslogD stop writing to the a log files and begin writing to the b log files. This will give you the opportunity to offload the a files and delete their current contents, making them ready for SyslogD use again by swapping the other way from the syslog.conf.b file to the syslog.conf.a file.

5.3 InetD Customization

InetD is used as the listener process for the TelnetD server, the REXECD server and the RSHD server from the TCP/IP for MVS OpenEdition Applications Feature.

The InetD configuration information is by default placed in /etc/inetd.conf. If you need to specify server run-time options for the servers that are started via InetD, you have to do so in the /etc/inetd.conf file. We describe the various options per server, when we later discuss each of the servers in detail.

As InetD opens sockets and issues listen() calls on these sockets, you need to pay attention to when you start InetD and when you need to restart InetD.

In our sample installation we chose not to start InetD until our default AF_INET transport provider had connected to OpenEdition. We started InetD via a small shell script for the same reasons as for SyslogD. The shell script to start InetD looks like the following:

```
#
# Start the INETD daemon
#

export _BPX_JOBNAME='INETD'
export _CEE_RUNOPTS='ALL31(ON)'
/usr/sbin/inetd /etc/inetd.conf &

echo -- /u/alfredc/restart_inetd.sh script executed, date
```

Please refer to 4.11, “Automation of OMVS Operations” on page 126 for details on the operational aspects of starting and stopping InetD.

The complete /etc/inetd.conf configuration file that was used in our sample setup looks like the following:

```
#======================================================================
# service | socket | protocol | wait/ | user | server  | server program
# name    | type   |          | nowait|      | program |   arguments
#======================================================================
#
telnet   stream tcp nowait OMVSKERN /usr/sbin/otelnetd otelnetd -l -m -D all -t
shell    stream tcp nowait OMVSKERN /usr/sbin/rshd rshd -d
login    stream tcp nowait OMVSKERN /usr/sbin/rlogind rlogind -m
exec     stream tcp nowait OMVSKERN /usr/sbin/rexecd rexecd -d
echo     stream tcp nowait OMVSKERN internal
discard  stream tcp nowait OMVSKERN internal
chargen  stream tcp nowait OMVSKERN internal
daytime  stream tcp nowait OMVSKERN internal
time     stream tcp nowait OMVSKERN internal
echo     dgram  udp wait   OMVSKERN internal
discard  dgram  udp wait   OMVSKERN internal
chargen  dgram  udp wait   OMVSKERN internal
daytime  dgram  udp wait   OMVSKERN internal
time     dgram  udp wait   OMVSKERN internal
```

In addition to the three servers from the TCP/IP for MVS OpenEdition Applications Feature, InetD also starts the RloginD server. InetD handles a range of servers internally without forking new processes with independent programs. These servers are the echo, discard, chargen, daytime and time servers.

5.4 TelnetD Customization

The TelnetD server is used to enable remote telnet clients to log into your OpenEdition shell environment in either raw-mode (also called character-at-a-time mode) or line-mode.

See Figure 100 on page 139 for an overview of how the TelnetD server is implemented in OpenEdition.

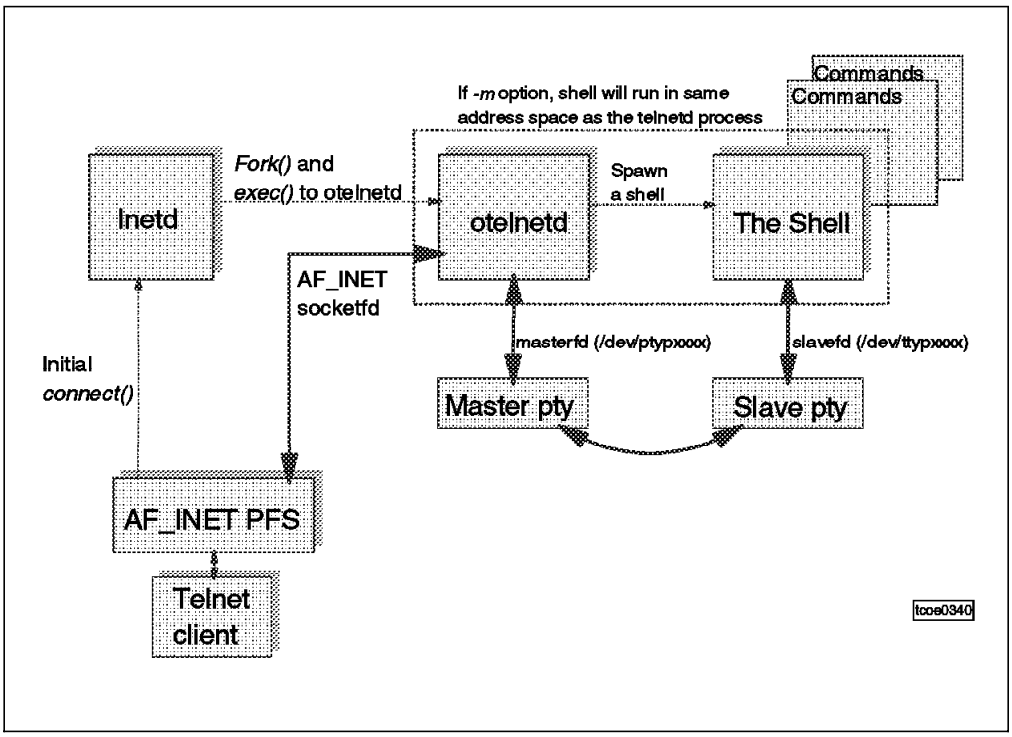


Figure 100. TelnetD Overview

ASCII/EBCDIC translation is done by the otelnetd process. The telnet client end user can specify which translation to use via the chcp shell command.

```
/u/alfredc: >chcp -a IBM-850 -e IBM-1047 1
/u/alfredc: >chcp -q 2
 ASCII code page : IBM-850
EBCDIC code page : IBM-1047
/u/alfredc: >
```

1 Sets the ASCII and EBCDIC code pages to be used.

2 Queries the current settings of code pages.

When a CHCP shell command is executed, the TelnetD process is informed about the code page change via urgent data over the pseudoterminal connection (the master/slave pty interface). A user may select to have a CHCP command executed as part of the user′s login process, for example, via the user′s $HOME/.profile or $HOME/.setup shell scripts.

5.4.1 Pseudoterminals

In preparing for the TelnetD server you need to ensure that during OpenEdition installation and customization you created a sufficient number of pseudoterminal files. The pseudoterminal files are created in the Hierarchical File System, in the /dev directory. The file names are ptypnnnn and ttypnnnn. where nnnn is a number from 0000 to 9999. These files are used in pairs by the TelnetD server and the RloginD server.

During OpenEdition installation you executed the sample REXX program BPXISMKD from SYS1.SAMPLIB to create a number of directories and files in your Hierarchical File System. One section of this REXX program creates 50

pseudoterminal pairs. You may have increased that by customizing the BPXISMKD REXX program before executing it. To verify how many pseudoterminal pairs you have defined in your system, list the /dev directory and locate the highest numbered ptypnnnn entry you find. If that is ptyp0050, and you want to increase the number of pseudoterminals to, for example, 75, you can either issue individual mknod commands (which probably is going to be a bit cumbersome), or you can create a small REXX program like the following:

```
/* REXX */
ptystart = 51    /*First new ptyp and ttyp number*/
ptylast = 75     /*New maximum number            */
call syscalls('ON')
Do pty = ptystart to ptylast
 ptyname = '/dev/ptyp'||right(pty,4,0)
 ttyname = '/dev/ttyp'||right(pty,4,0)
 say 'Creating 'ptyname' and 'ttyname
 address syscall "mknod" ptyname "666 1 0"
 address syscall "mknod" ttyname "666 2 0"
End
exit(0)
```

The above REXX program will increase your pseudoterminal pairs from 50 to 75. Please note that the REXX program must be executed from a superuser.

After you have increased the number of pseudoterminals in your Hierarchical File System, you need to let OpenEdition know that more pseudoterminals now are available. You do so by updating the MAXPTYS statement in your BPXPRMxx parmlib member:

```
MAXPTYS(76)
```

The MAXPTYS value is specified as one above the highest pseudoterminal number you created. In the above setup, you have from pseudoterminal 0000 to pseudoterminal 0075, which gives 76 pairs in total.

5.4.2 Starting the TelnetD Server

The TelnetD server is started via InetD.

The default port number for the TelnetD server is 23. Although this is a well-known port number, it must be reserved to OpenEdition in the PROFILE data set of TCP/IP for MVS:

```
23 TCP OMVS ;  OE telnet server
```

If the default port number 23 is used, a client has to know only the name or IP address of the host to establish a connection and can use a command such as:

```
telnet -t vt220 9.24.104.126
```

It is also possible to assign a different port to the OE telnet server in the PROFILE data set of TCP/IP for MVS, for example:

```
1223 TCP OMVS ;  OE telnet server
```

In this case, a client has to specify both the host name (or IP address) and the port number of the OE telnet server with the telnet login command:

```
telnet -t vt220 9.24.104.126 1223
```

If you assign an alternate port number to your OpenEdition TelnetD server, you also need to update your OE resolver /etc/services configuration file with the chosen port number in order for InetD to listen for telnet client requests on the chosen port number:

```
telnet        1223/tcp
```

If your configuration has more than one TCP/IP for MVS stack connected to OpenEdition, all of these stacks must have identical port reservations for the OE telnet server. The chosen port number is of systemwide value in the OpenEdition environment.

A TelnetD server process is forked from InetD whenever a telnet client connects to OpenEdition. See Figure 100 on page 139 for an overview of how TelnetD operates in the OpenEdition environment.

The TelnetD server supports the following server options to be specified in the /etc/inetd.conf file:

C Output messages will be in uppercase letters.
D Various debugging options.
h Prevent display of host-specific banner page (from /etc/banner).
k Prevent raw-mode initialization.
l Set default mode for login to line mode.
m Run all forked or spawned processes in the same address space, recommended for performance improvement.
n Disable TCP keep-alives.
t Activate internal tracing information.
U Refuse connections from any address that cannot be resolved into a host name via a gethostbyaddr() call.

Note: Please note that the configuration options are case sensitive.

Any of these parameters can be changed dynamically by modifying the configuration file /etc/inetd.conf and issuing a kill -SIGHUP command to force InetD to re-read the configuration file. All new telnet connections that are established after the configuration file has been re-read will use the new options.

An environment-specific banner page can be created in the file /etc/banner using standard edit functions. It will be displayed at the client workstation after the user ID and password have been entered.

The following is a sample banner page that was used in our setup:

```
**************************************************************************
*                                                                        *
*  Welcome to OpenEdition MVS 5.2.2 on ITSO Raleigh MVS18OE              *
*                                                                        *
*  This is the /etc/banner page for telnet server use.                   *
*                                                                        *
*  Need information on how to use OpenEdition MVS?  Contact              *
*  your local ITSO MVS system administrator.                             *
*                                                                        *
**************************************************************************
```

For a detailed description of differences between the standard TCP/IP for MVS telnet server and the OE TelnetD server, see *TCP/IP for OpenEdition MVS: Applications Feature Guide*, SC31-8069 - Appendix B.

All messages from the TelnetD server are logged by SyslogD according to the SyslogD configuration file settings. The TelnetD server uses a facility name of local1 for all its messages.

5.4.3 Termcap And Terminfo

For telnet sessions, you may need some definitions of terminal capabilities in order to let OpenEdition application programs manipulate the terminal output correctly. These definitions are kept in the terminfo database.

A UNIX system, as other operating systems, must have at least one terminal attached to it. In very early days these were typewriters having keyboard input and a paper output. These terminals were later replaced by video display units (VDU) which actually behave like the typewriter did earlier; the paper is just replaced by the screen. All VDUs understand a data stream, which contains ASCII characters to be printed, including control characters, such as carriage return, or new line. Differences showed up when individual manufacturers started to add specific commands to their terminals. Specific commands may be cursor up, and cursor left, to name a few.

As long as one uses simple printing commands such as `cat`, there is no obvious difference among all terminals. But these differences have to be taken into account as soon as one runs a program that uses special commands. This is mostly the case with editors, such as vi, or emacs. These programs need to know, for example, how to move the cursor to a specific location.

As usual, there is more than one approach to solve this problem. The first is to write specific drivers for each terminal. Another solution is to create a database that contains definitions of all capabilities each terminal has and a subroutine library, which gives access to the database.

The first approach would have been a typical UNIX solution. Writing a device driver is not an unusual task in this environment. But, the latter solution was chosen and is today a somewhat agreed-upon standard in the world of UNIX. This is what the termcap database provides. It is actually a readable text file. It has its roots in the Berkley Software Distribution (BSD). In UNIX systems that have their roots in System V UNIX, the equivalent implementation of termcap is the terminfo database. The most important difference is that terminfo is a compiled database. For that reason you need a compiler (tic) to create the terminfo database. While termcap contains all definitions in one file, terminfo organizes the information into a directory structure (see Figure 102 on page 143).

To create the terminal information database, you need to run the *tic* compiler as shown in Figure 101.

```
tic /usr/share/lib/terminfo/ibm.ti
tic /usr/share/lib/terminfo/dec.ti
tic /usr/share/lib/terminfo/wyse.ti
```

Figure 101. Creating the Terminfo Database

We found it necessary to run the tic command with all three .ti files that are provided by OpenEdition. To create the directory structure as indicated in Figure 102 on page 143, we set the environment variable TERMINFO=/usr/Terminfo before running the tic commands. The TERMINFO

environment variable must also be set for every user that logs into the OpenEdition shell in order for the application programs that are executed in the shell environment to locate the terminfo database. Whenever a shell environment is created, the terminal type of the client user is registered in the environment variable TERM. The TSO OMVS command environment handles TERM as if it were set to TERM=dumb. In a telnet or rlogin session the TERM environment variable is set to the terminal type that the telnet client uses or emulates. In our case we used an OS/2 telnet client with a terminal type specification of vt220, which results in a TERM=vt220 setting in the shell.

```
pwd=/usr/Terminfo: >ls -l
total 0
drwxr-xr-x   2 HDM      OMVSGRP        0 Apr 24 23:13 a
drwxr-xr-x   2 HDM      OMVSGRP        0 Apr 24 23:13 c
drwxr-xr-x   2 HDM      OMVSGRP        0 Apr 24 23:13 d
drwxr-xr-x   2 HDM      OMVSGRP        0 Apr 24 23:13 h
drwxr-xr-x   2 HDM      OMVSGRP        0 Apr 24 23:16 i
drwxr-xr-x   2 HDM      OMVSGRP        0 Apr 24 23:13 j
drwxr-xr-x   2 HDM      OMVSGRP        0 Apr 24 23:13 l
drwxr-xr-x   2 HDM      OMVSGRP        0 Apr 24 23:13 L
drwxr-xr-x   2 HDM      OMVSGRP        0 Apr 24 23:13 v
drwxr-xr-x   2 HDM      OMVSGRP        0 Apr 24 23:13 w
drwxr-xr-x   2 HDM      OMVSGRP        0 Apr 24 23:12 x
pwd=/usr/Terminfo: >
```

Figure 102. Directory Structure Created by Tic

After we had installed the complete terminfo structure we were able to run full-screen applications, such as man or pg, without any problems. Table 10 shows the contents of the /usr/Terminfo directory structure.

Table 10. Terminfo Directory Structure and Contents

Directory	Terminal Definition Files
a	aixterm, aixterm-m, aixterm-m-old, aixterm-old
c	cdef
d	dumb
h	hft, hft-c-old, hft-m-old, hft-nam-old, hft-c, hft-m, hft-nam, hft-old
i	ibm3101, ibm3152, ibm3163, ibm5550, ibm6155, ibm8515, ibm3151, ibm3152-132, ibm3164, ibm5570, ibm6155-113 ibm8604, ibm3151-132, ibm3152-25, ibm5081, ibm6153, ibm6155-56, ibmpc, ibm3151-25, ibm3152-PS2 ibm5081-113 ibm6153-40 ibm8503, ibmpcc, ibm3151-51, ibm3161, ibm5081-56, ibm6153-90, ibm8507, ibmpdp, ibm3151-61, ibm3161-C, ibm5081-old ibm6154 ibm8512 ibm3151-noc, ibm3162, ibm5151, ibm6154-40, ibm8513, ibm3151-S, ibm3162-132, ibm5154, ibm6154-90, ibm8514
j	jaixterm, jaixterm-m
l	lft, lft-pc850
L	LFT, LFT-PC850
v	vs100, vt100, vt100-nam, vt220, vt320, vt330, vt340, vs100s, vt100-am, vt100x, vt220-8, vt320-8, vt330-8
w	wy60-316X, wy60-AT, wy60-PC, wyse60-316X, wyse60-AT, wyse60-PC
x	xterm, xterms

Users who use telnet to enter their OpenEdition systems should have a TERMINFO environment variable setting in their $HOME/.setup shell script:

```
TERMINFO="/usr/Terminfo"
export TERMINFO
```

If you are interested in more information on termcap and terminfo, refer to *Termcap & Terminfo*, published by O'Reilly & Associates.

5.4.4 Curses

When discussing the termcap and terminfo definitions one is immediately lead into the subject of curses. Curses is a set of approximately 300 functions that allows an application program to manipulate the screen contents directly. Manipulation may be made either to the readable contents such as text or to the appearance of the screen (fields, windows, colors, etc). These functions were added when the SPEC1170 definitions were made, and they are implemented in OpenEdition. Implementation of applications that use curses require that the terminfo database has been set up correctly, as we discussed in 5.4.3, "Termcap And Terminfo" on page 142.

As an example of a very simple curses application, we retrieved some sample source code from the O'Reilly ftp server. The program is really not a sophisticated curses application but it allows one to test some basic curses functions. For your reference, we have included the source code in Appendix C, "Sample Curses Application" on page 221.

5.5 REXECD and RSHD Customization

Both the REXECD server and the RSHD server are used to execute OpenEdition shell commands from remote users. See Figure 103 for an overview of how both the REXECD server and the RSHD server are implemented in the OpenEdition environment.

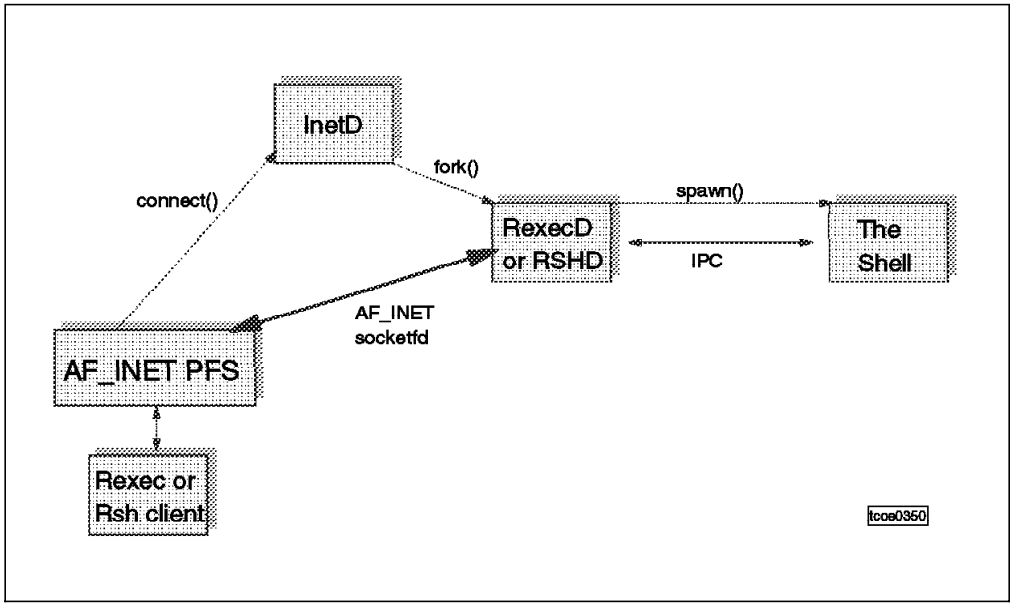


Figure 103. REXECD and RSHD Implementation Overview

The REXEC protocol requires a remote user to explicitly enter both a user ID and a password on the REXEC command line invocation:

```
rexec mvs18o -l alfredc -p password ls -al
```

The RSH protocol does not require an end user to enter a user ID and a password. If the user does not enter a user ID, the local user ID will be sent to the remote RSHD server as the remote user ID, but the protocol and RSH command syntax does not support a password parameter, such as the REXEC command does. Some RSHD servers implement an rhosts.data file on the RSHD server host to authorize certain client users on certain remote hosts to execute commands on the RSHD server without authentication in terms of verifying a password. This implementation has, for security reasons, not been ported to the OpenEdition RSHD server, and the RSHD server in OpenEdition does *not* process any remote commands without a valid MVS user ID and password.

So how does the RSH client user then enter a command? The syntax is as follows:

```
rsh mvs18o -l alfredc/password ls -al
```

On the -l parameter, the end user has to enter both an MVS user ID and the password with the user ID and password separated by a slash (/).

5.5.1 Starting the REXECD and RSHD Servers

Both the REXECD and the RSHD servers are started from InetD.

The default port numbers for the two servers are 512 and 514, and both must be reserved in the PROFILE data sets of the TCP/IP stacks that act as OpenEdition AF_INET transport providers:

```
PORT
   512 TCP OMVS                ; OE Remote Execution Server
   514 TCP OMVS                ; OE Remote Shell Server
```

Most REXEC and RSH client implementations do not allow the end user to specify an alternate port number, so we do not recommend that you try to start these servers on alternate port numbers.

The REXECD server accepts the following arguments in the /etc/inetd.conf file:

d Write debug information to SyslogD, facility name daemon.
C Uppercase translate all output messages.
L Write each successful login to SyslogD with user ID and command content, facility name auth.
V Write name and PTF level information to SyslogD, facility name daemon.

The RSHD server accepts the following arguments in the /etc/inetd.conf file:

d Write debug information to SyslogD, facility name daemon.
C Uppercase translate all output messages.
L Write each successful login to SyslogD with user ID and command content, facility name auth.
V Write name and PTF level information to SyslogD, facility name daemon.
a Double-check the client IP address and host name correlation. The RSHD server will first do a gethostbyaddr() to obtain the host name of the client host. It will then do a gethostbyname() and verify that IP address of the client is the same as that returned from the name server.

Note: If you enable the RSHD server L or d option, the entry in SyslogD will include both user ID and password in clear text. Either do not specify these flags, or make sure your SyslogD log files are properly protected.

The RSHD server will not execute a command if the client host IP address can not be resolved to a host name.

There are a few situations where the RSHD server may encounter an error so early in the processing of a command that the server has not established a proper EBCDIC-to-ASCII translation yet. In such a situation, the client end user may see “garbage” data returned to his or her terminal. A packet trace will reveal that the response is in fact returned in EBCDIC, which is the reason for the garbage look on an ASCII workstation. We have seen this happen if the OpenEdition name resolution has not been configured correctly, so the RSHD server, for example, was not able to resolve IP addresses and host names correctly. If your RSH clients encounter such a problem, please go back and check your name resolution setup. If you are using a local hosts table, make sure that the syntax of the entries in your hosts file is correct.

5.6 REXEC Client

The REXEC client component of the TCP/IP for MVS OpenEdition Applications Feature allows you, from the OpenEdition shell environment, to execute commands on remote hosts.

The syntax of the REXEC client command is as shown in Figure 104.

```
Usage: orexec -V -d -l <user> -p <pwd>
              -s <port> fhost command
       options: -
                 -?        display this message
                 -d        turn on debug tracing
                 -l <usr>  specifies remote login id
                 -p <pwd>  specifies remote password
                 -s <port> specifies server port
                 -C        Uppercase messages
                 -V        display APAR level

Example: orexec -d -l guest -p guest hostname ls -l
```

Figure 104. REXEC Client Command Syntax in OpenEdition

The REXEC command in OpenEdition is orexec.

A sample invocation of the orexec command from OpenEdition to a REXECD server on an OS/2 host looks like the following:

```
/u/alfredc: >orexec -l alfredc -p nosecret alfred dir config.sys

The volume label in drive C is C_DRIVE.
The Volume Serial Number is E6C3:0014.
Directory of C:\

 5-20-96   1:17p      5138           0  CONFIG.SYS
        1 file(s)       5138 bytes used
                     2591744 bytes free
/u/alfredc: >
```

5.7 FTPD Customization

The OE FTPD server is a full-function MVS FTP server, which allows you to transfer both standard MVS data sets and Hierarchical File System files in and out of an OpenEdition system. The OE FTPD server is based on the C-FTP server from TCP/IP Version 3 Release 1 for MVS. All the functions you find in the C-FTP server from TCP/IP Version 3 Release 1 for MVS are included in the OE FTPD server, and in addition to that, the OE FTPD server supports Hierarchical File System files. The fact that the server works with both standard MVS data sets and Hierarchical File System files also means that functions from the C-FTP server, such as JES file type and SQL file type functions, are supported by the OE FTPD server for both standard MVS data sets and Hierarchical File System files. You can run SQL queries based on SQL statements in Hierarchical File System files, or you can submit MVS batch jobs based on JCL data in a file in your Hierarchical File System.

The FTPD server does not use InetD as a listener process, but has its own listener program. Please see Figure 105 for an overview of the FTPD server implementation in OpenEdition.

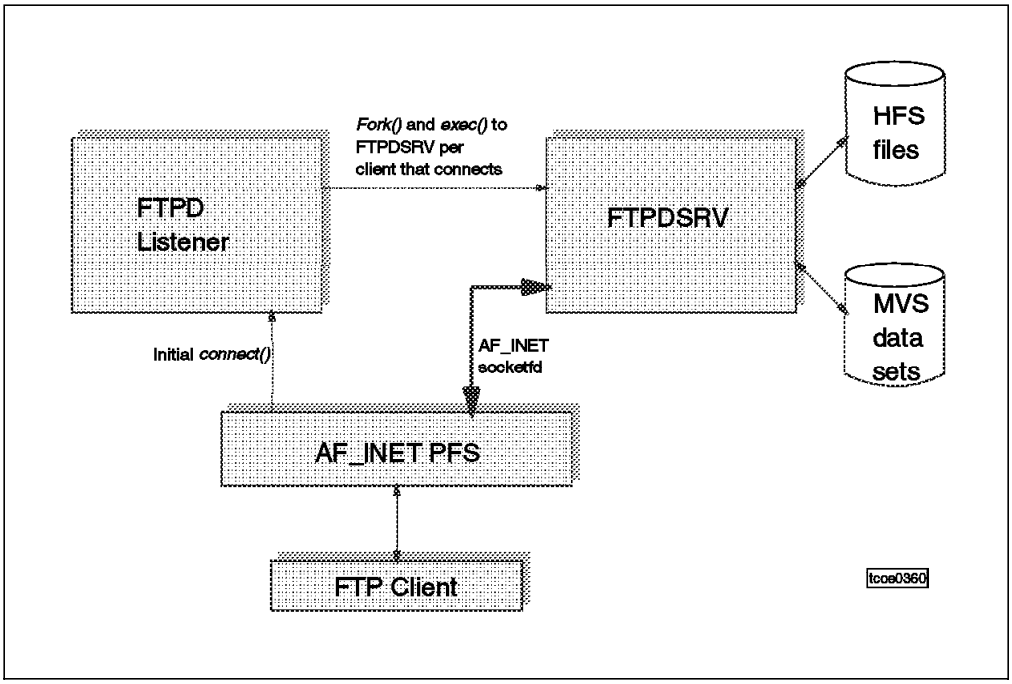


Figure 105. FTPD Implementation Overview

5.7.1 Starting the FTPD Server Listener Program

The FTPD server requires, as do other servers in OpenEdition, the ability to change the user security environment to that of the client FTP user. The FTPD server listener process must therefore be started under a user ID with a UID=0 and a permission to use the BPX.DAEMON facility class resource. In our sample setup, we used the usual OMVSKERN user ID for this purpose.

The FTPD listener program is called ftpd and the program that processes each FTP client connection is called the ftpdsrv program. During installation of the TCP/IP for MVS OpenEdition Applications Feature, both of these programs were installed into the /usr/sbin directory with the sticky bit set. The installation process also placed a copy of these two programs into your TCP/IP for MVS SEZALINK library.

The FTPD listener can be started in more ways, from a shell script, via a BPXBATCH job or as a POSIX(ON) program. We decided to start it as a POSIX(ON) program because we wanted to control by means of the SYSFTPD DD-statement which FTP.DATA configuration data set the FTPD server used. If we only started one FTPD server in OpenEdition, we could have used the default FTP.DATA location in the Hierarchical File System, the /etc/ftp.data file, but we wanted to start two instances of the OE FTPD server and needed to control the function of each of these servers via different FTP.DATA configuration options, so we chose to use the SYSFTPD DD-statement approach for this purpose. Such a technique will not work, as you may recall from earlier discussions, if the initial FTPD process is started in a forked process as it would be if we started it from the interactive shell environment or from a BPXBATCH invocation.

The JCL we used to start our normal OE FTPD server looks like the following:

```
//FTPD  1  PROC
//*
//*  OpenEdition MVS FTP Server main process
//*  Resulting address space name will be FTPD1, when
//*  we use this method to start FTPD
//*
//FTPD  EXEC PGM=FTPD,REGION=40M,  2
//           PARM='POSIX(ON),ALL31(ON)/'
//*
//* FTPD is in SYS1.TCPIP.SEZALINK (on the LINKLST)
//* FTCHKxxx routines are in the STEPLIB load library
//*
//STEPLIB DD DSN=SYS1.TCPIP.ITSC.LINKLIB,DISP=SHR  3
//SYSFTPD DD DSN=TCPIP.ITSC.CNTL(OEFTPSTD),DISP=SHR  4
```

1 When we use this technique to start the FTPD listener, the resulting listener process will get a job name of this procedure name suffixed with 1 - FTPD1. The address space you start with an MVS operator start command, the FTPD address space, starts the FTPD listener program, which does some initialization (including reading SYSFTPD) before it forks another address space, the FTPD1 address space. The FTPD1 address space is the address space in which the FTPD listener program will execute until closed down by an MVS operator STOP command (P FTPD1) or an OpenEdition kill signal.

Do not be alarmed when you see that the address space you just started stops a few seconds after you have started it. Everything is OK. The FTPD address space started up with the ftpd program. The ftpd program read the FTP.DATA configuration options, built an in-storage character string that represented the

FTP.DATA options and passed this string on to the forked copy of the ftpd program. Why did it do this? Why couldn't the forked address space just read SYSFTPD? Remember our discussion about DD-name allocations across a fork? The forked FTPD1 address space does not have access to that DD-name allocation, which is the reason why the ftpd program uses this technique to make the FTP.DATA options available to the FTPD listener process. The listener process in turn uses the same technique to pass them on to the ftpdsrv program in the address spaces it forks, when FTP clients connect.

2 The FTPD server listener program is called ftpd, and will be executed from an MVS library on the system link list, the TCP/IP for MVS SEZALINK library.

3 If you have developed your own FTPD security exit routines (FTCHKIP, FTCHKJES, FTCHKPWD, FTCHKCMD) or SMF exit routine (FTPSMFEX), they must be either in a library on your system link list, or you must add a STEPLIB library to this JCL procedure. As mentioned earlier, a STEPLIB allocation is the only DD-name allocation that will be passed to forked child processes.

Notes:

1. Please remember that the library must be RACF program controlled (see 2.1, "Program Control" on page 50) and APF authorized. The FTPD server, in addition to using OpenEdition functions to change the security environment, also uses standard MVS functions to check a user's authorization to standard MVS resources before a user is allowed to transfer MVS data sets in or out of your MVS system.
2. If you enable SQL query support, DB2 load modules are going to be loaded from your DB2 DSNLOAD library and most likely also from your DB2 DSNEXIT library. These libraries must be under RACF program control too, in order for the FTPD server address space not to become dirty when the DB2 load modules are loaded.

```
RALTER  PROGRAM * ADDMEM('db2.version.DSNEXIT'/volser/NOPADCHK) UACC(READ)
RALTER  PROGRAM * ADDMEM('db2.version.DSNLOAD'/volser/NOPADCHK) UACC(READ)
```

4 In our sample setup, we use the SYSFTPD DD-statement to direct the FTPD server to its FTP.DATA configuration options. The search order for FTP.DATA is the same as for the non-OE C-FTP server with one exception: a search for /etc/ftp.data has been added as the first place to look for an FTP.DATA configuration file:

1. /etc/ftpd.data
2. A SYSFTPD DD-name allocation
3. *jobname*.FTP.DATA or *userid*.FTP.DATA
4. SYS1.TCPPARMS(FTPDATA)
5. *datasetprefix*.FTP.DATA

The search step for FTP.DATA that uses a job name as high-level qualifier will use the original jobname (the FTPD name in our setup) and not the FTPD1 jobname. This original jobname will be used for search for both FTP.DATA and translation tables: FTPD.FTP.DATA and FTPD.STANDARD.TCPXLBIN. If the ftpd listener program were started from the shell, these search steps would use the user ID of the process that started the listener program. The *datasetprefix* used

in the last search step comes from the OE resolver configuration file, in our setup /etc/resolv.conf.

When a new FTP client connects to the FTPD listener process, the FTPD listener forks another address space that uses the exec() services to start the connection specific server program, the ftpdsrv program.

The FTP.DATA configuration options we used for this FTPD server are to a large extent similar to the ones you might have used for the C-FTP server in the TCP/IP Version 3 Release 1 for MVS product, with some new options that are specific to the OE FTPD server:

```
;*********************************************************************
;                                                                    *
;   Name of File:      TCPIP.ITSC.CNTL(OEFTPSTD)                     *
;                                                                    *
;   FTP.DATA for standard OE FTP server on Ports 20/21               *
;                                                                    *
;*********************************************************************
Primary         1          ; Primary allocation is 1 track
Secondary       20         ; Secondary allocation is 20 tracks
Directory       15         ; PDS allocated with 15 directory blocks
Lrecl           255        ; Logical Record Length of 255
BlockSize       6144       ; Block Size of 6144
AutoRecall      true       ; Migrated HSM files recalled automatically
AutoMount       false      ; Non-mounted volumes not mounted auto.
DirectoryMode   false      ; Use all qualifiers (Datasetmode)
Volume          WTLTCP     ; Volume serial number for new data sets
SpaceType       TRACK      ; Data sets allocated in tracks
Recfm           VB         ; Variable Blocked record format
Filetype        SEQ        ; File Type = SEQ (default)
Smf             170        ; The SMF record sub type to be used
Mgmtclass       TCPMGMT    ; SMS management class for new data sets
Ctrlconn        IBM-850    ; ASCII code page for control connection
Sbdataconn      (IBM-1047,IBM-850) ; EBCDIC, ASCII code page data conn.
Replylanguage   (English,C)    ; English replies (use as default)
Replylanguage   (Dutch,NL_NL)  ; Dutch replies (does NOT work in V3R1)
Replylanguage   (Danish,DK_DK) ; Danish replies (does NOT work in V3R1)
Quotesiverride  TRUE       ; Single quotes means qualified name
Umask           027        ; Make new HFS files rw- r-- ---
```

5.7.2 Users of the OE FTPD Server

The OE FTPD server uses the home directory specification in a user's RACF profile as the default current directory when a user logs on to the server.

Note: Please note that this is a change from the non-OE C-FTP server. The non-OE C-FTP server uses the PREFIX value from a user's TSO profile.

If your FTP clients want to transfer MVS data sets in or out of the OE FTPD server, they can change the current working directory after they have logged in:

```
[C:\]ftp mvs18o
IBM TCP/IP for OS/2 - FTP Client ver 15:51:28 on Nov 19 1994
Connected to mvs18o.itso.ral.ibm.com.
220-FTPD1 IBM MVS V3R1 at mvs18oe.itso.ral.ibm.com, 17:17:42 on 1996-05-28.
220 Connection will close if idle for more than 5 minutes.
Name (mvs18o): alfredc
331 Send password please.
Password: .......
230 ALFREDC is logged on.  Working directory is "/u/alfredc".  1
ftp> cd 'alfredc'  2
250 "ALFREDC." is the working directory name prefix.
ftp>
```

1 The default working directory just after having logged in is the user's HOME directory.

2 Change working directory to an MVS data set high-level qualifier.

A user of the OE FTPD server must have a valid OMVS identity in terms of an OMVS segment in the user's RACF user profile in order just to log on to the OE FTPD server. This is the case even if the user does not intend to transfer files in or out of the Hierarchical File System, but just want to use the server for transferring standard MVS data sets. If you have users in your environment who do not have an OMVS segment in their RACF user profiles and you want to enable these users to transfer MVS data sets in and out of your MVS system, you need to start the non-OE FTPD server for this use. The best setup for this purpose will be a separate stack TCP/IP for MVS configuration as described in 4.4, "TCP/IP for MVS, Separate Stacks" on page 92 with the non-OE FTPD server running on the non-OE TCP/IP for MVS stack and the OE FTPD server running in OpenEdition using the OE TCP/IP for MVS stack.

If FTP clients receive error messages like the following, when they try to use your OE FTPD server, they most likely do not have a valid OMVS segment in their RACF user profiles.

```
[C:\]ftp mvs18o
IBM TCP/IP for OS/2 - FTP Client ver 15:51:28 on Nov 19 1994
Connected to mvs18o.itso.ral.ibm.com.
220-FTPD1 IBM MVS V3R1 at mvs18oe.itso.ral.ibm.com, 13:45:23 on 1996-05-28
220 Connection will close if idle for more than 5 minutes.
Name (mvs18o): alfred
331 Send password please.
Password: ......
550 PASS command failed - getpwnam() error : EDC5163I SAF/RACF extract error.
Login failed.
ftp>
```

5.7.3 Data Set Name or File Name

The OE FTPD server determines for each resource you ask it to work with if the resource is an MVS data set or a Hierarchical File System file.

It does so based on an analysis of the resource name. You do not have to specify explicitly if you are working with MVS data sets or Hierarchical File System files.

- Is the name a fully qualified resource name (enclosed in single quotes)?

Use the name as it is. If the name begins with a slash (/), then treat it as an HFS file name. Otherwise treat it as an MVS data set name.

  - ′mydata.set′ - MVS data set MYDATA.SET
  - ′/u/user/myfile′ - HFS file /u/user/myfile

- Is the name not a fully qualified resource name?

  If the name begins with a slash, use the name as it is and treat it as an HFS file name.

  - /u/user/yourfile - HFS file /u/user/yourfile

  If the name does not begin with a slash, then append the value of the current working directory to the beginning of the name and treat the combined name as a fully qualified name.

  - myfile - if current working directory is /u/user, the resulting name is an HFS file called /u/user/myfile. If the current working directory is myuserid, the resulting name is an MVS data set called MYUSERID.MYFILE.

MVS data set names are translated to uppercase before the FTPD server accesses the data sets. HFS file names are left as is, because mixed case file names are fully valid in the Hierarchical File System.

File names in the Hierarchical File System may consist of any characters, including quotes and spaces. This poses a small dilemma if, for example, you want to transfer a file in the HFS that has a name of /u/user/c/′weird.name′, which is a fully valid name in the Hierarchical File System.

Assume the following sequence of events:

1. Current working directory is /u/user/c
2. User enters get ′weird.name′

According to our rules described earlier, this is a fully qualified MVS data set name, but that is not what the user meant. The OE FTPD server has a new option available to deal with this dilemma: the quotes override option. It can be set to an installation-wide default value in your FTP.DATA configuration and it can be changed by individual users via a SITE (NO)QUOTESOVERRIDE command.

Quotes override refers to overriding the current directory specification in this situation or not.

If QUOTESOVERRIDE=TRUE, the above situation would result in the MVS data set ′WEIRD.NAME′ being transferred.

If QUOTESOVERRIDE=FALSE, the above situation would result in the HFS file /u/user/c/′weird.name′ being transferred.

Another interesting file name in the Hierarchical File System could be the following: ••very•weird′Name••. (The • in the name stands for a blank character.) This file name starts with two blanks, has a blank and a quote in the middle of the name and ends with two blanks. Although special, it is a valid HFS file name.

The OE FTPD server deals with such a name based on the FTP protocol subcommand contents. If a user wants to retrieve the above file, the FTP subcommand would be a RETR subcommand. The exact contents of the subcommand would be as follows:

```
RETR   very weird'Name  <eol>
```

The RETR command is followed by *one* blank before the file name starts. In this example, there are three blanks following the RETR command, which means that the file name starts with two blanks. Everything up until the <eol> control character is considered to be the resource name, including the two trailing blanks in the above example. The resource name is then treated as described earlier. If the current working directory is an MVS data set high-level qualifier, the result would be a nonvalid MVS data set name, but if the current working directory is a directory path in the Hierarchical File System, the resulting name would be a valid file name.

Note: Please note that not all FTP clients will be able to deal with such file names, especially not file names with leading or trailing blanks.

When you work with file names in the Hierarchical File System, there are a couple of character sequences that have special meaning:

~/ (tilde slash) This stands for your $HOME directory.

../ (dot dot slash) This means back up one directory level.

These may be combined in a resource name. Consider the following example: your current working directory is /u/dilbert/c/trojan_horses and your $HOME directory is /u/weirdo.

- You enter `get ~/mysweetie` - resulting file name becomes: /u/weirdo/mysweetie.
- You enter `get ../../bin/sweeproot` - resulting file name becomes: /u/dilbert/bin/sweeproot.
- You enter `get ~/../boss/salary_list` - resulting file name becomes: /u/boss/salary_list.

5.7.4 New SITE Options

The OE FTPD server has introduced a few new SITE options and made a single old one obsolete.

The XLATE SITE command has been removed and replaced by two new SITE commands: the CTRLCONN and the SBDATACONN options.

The following SITE options are new:

CHMOD

Change the mode bits of a file in the Hierarchical File System.

The standard chmod shell command syntax is supported. If you use the CHMOD subcommand in a SITE command, it must be either the only subcommand in that SITE command or it must be the last subcommand.

CTRLCONN

Set the ASCII/EBCDIC translation option for the FTP control connection.

The default translation for the control connection is established via the new FTP.DATA keyword CTRLCONN.

You decide which ASCII code page to use. The FTPD server uses the server locale EBCDIC code page for the control connection.

LANG

Although documented, this SITE command has no meaning. Only English replies are available.

QLANG

Although documented, this SITE command has no meaning. Only English replies are available.

QUOTESOVERRIDE

Set the quotes override option to true (see 5.7.3, “Data Set Name or File Name” on page 151).

NOQUOTESOVERRIDE

Sets the quotes override option to false (see 5.7.3, “Data Set Name or File Name” on page 151).

SBDATACONN

Set the ASCII/EBCDIC translation option for the FTP data connection.

The default translation for the data connection is set via the SBDATACONN keyword in your FTP.DATA configuration data set.

You specify both an ASCII code page and an EBCDIC code page. If, for example, you want to use EBCDIC code page IBM-037 and ASCII code page IBM-850, you specify the following:

```
site sbdataconn=(IBM-037,IBM-850)
```

The code pages that are supported by the OE FTPD server are those that are supported by the OpenEdition iconv() service. Please refer to *IBM C/C++ for MVS/ESA C/MVS Programming Guide*, SC08-2062 for details on the supported code page conversions.

You may also use the translate table data sets from the non-OE C-FTP server. You do so by using the following syntax on your SITE command:

```
SITE sbdataconn=tcpip.omvs.german.tcpxlbin
```

Please note that you specify the fully qualified MVS data set name and not, as with the obsolete XLATE SITE command, a DD-name. You may also place your translate tables in the Hierarchical File System and refer to a Hierarchical File System file name in a SITE SBDATACONN command:

```
SITE sbdataconn=/u/userx/swedish.tcpxlbin
```

UMASK

UMASK is used to modify the initial permission bits of new files or directories in the Hierarchical File System. The value you specify for the UMASK is 3 octal digits. When a new file is created, OpenEdition uses a default set of permission bits of 666. When a new directory is created, the default set of permission bits is 777. You use the UMASK to control which of these default bits should be turned off. If you specify a UMASK of 077, then a new file will be created with permission bits of 600 and a new directory with 700.

```
Initial permission bits: 110 110 110 (666 = rw- rw- rw-)
UMASK:                   000 111 111 (077 = --- rwx rwx)
                         -------------------------------
Resulting permission:    110 000 000 (600 = rw- --- ---)
```

See Table 11 for an overview of how the new SITE commands correlate to new FTP.DATA options.

Table 11. FTPD New SITE Commands and FTP.DATA Options

SITE Command	FTP.DATA Option	Default Setting
CHMOD	No corresponding parameter	No defaults
CTRLCONN	CTRLCONN	Standard TCP/IP for MVS translation table data set. Search order is: 1. *jobname*.SRVRFTP.TCPXLBIN 2. *datasetprefix*.SRVRFTP.TCPXLBIN 3. *jobname*.STANDARD.TCPXLBIN 4. *datasetprefix*.STANDARD.TCPXLBIN If no translation table data set is found, then the control connection translation defaults to 7BIT.
LANG	No meaning, only English reply languages are supported.	Default is the C locale and the corresponding reply message catalog is found in /usr/lib/nls/msg/C/ftpdrply.cat
QLANG	No meaning, only English reply languages are supported.	
QUOTESOVERRIDE	QUOTESOVERRIDE TRUE	TRUE
NOQUOTESOVERRIDE	QUOTESOVERRIDE FALSE	TRUE
SBDATACONN	SBDATACONN	Standard TCP/IP for MVS translation table data sets. Search order is: 1. SYSFTSX DD-name 2. *jobname*.SRVRFTP.TCPXLBIN 3. *datasetprefix*.SRVRFTP.TCPXLBIN 4. *jobname*.STANDARD.TCPXLBIN 5. *datasetprefix*.STANDARD.TCPXLBIN
UMASK	UMASK	027

5.7.5 Running More FTPD Servers

You can start more instances of the FTPD server, each instance running on a separate set of port numbers and using a separate set of FTP.DATA options.

To illustrate this setup, we created a second FTPD server that was configured so it would, by default, process all get requests as SQL queries.

The JCL procedure to start this server looks like the following:

```
//FTPDSQL PROC
//*
//*  OpenEdition MVS FTP SQL Query Server main process
//*
//FTPD  EXEC PGM=FTPD,REGION=40M,
//           PARM='POSIX(ON),ALL31(ON)/PORT 2021'  1
//*
//* FTPD is in SYS1.TCPIP.SEZALINK (on the LINKLST)
```

```
//* FTCHKxxx routies are in the following load library
//* DB2 modules are needed for SQL support
//*
//* NB: All steplib libraries MUST be RACF PROGRAM controlled!
//*
//STEPLIB DD DSN=SYS1.TCPIP.ITSC.LINKLIB,DISP=SHR
//        DD DSN=SYS1.DSN230.DSNEXIT,DISP=SHR 2
//        DD DSN=SYS1.DSN230.DSNLOAD,DISP=SHR
//SYSFTPD DD DSN=TCPIP.ITSC.CNTL(OEFTPSQL),DISP=SHR
```

1 This server starts on an alternate control port, port number 2021 and it will use 2020 as data port. These port numbers should be reserved to OMVS in your TCP/IP for MVS PROFILE data set:

```
PORT
  2020 TCP  OMVS                  ; OE FTP Alternate Server data port
  2021 TCP  OMVS                  ; OE FTP Alternate Server control port
```

2 As this server is used for SQL queries, we add the DB2 load libraries to our STEPLIB. If your DB2 load libraries are on your system link list, you do not need to add them to STEPLIB. Please remember to RACF program control your DB2 load libraries.

The FTP.DATA options used for this server look like the following:

```
;*********************************************************************
;                                                                    *
;   Name of File:      TCPIP.ITSC.CNTL(OEFTPSQL)                     *
;                                                                    *
;   FTP.DATA for SQL OE FTP server on ports 2020/2021                *
;                                                                    *
;*********************************************************************
Primary         1          ; Primary allocation is 1 track
Secondary       20         ; Secondary allocation is 20 tracks
Directory       15         ; PDS allocated with 15 directory blocks
Lrecl           255        ; Logical Record Length of 255
BlockSize       6144       ; Block Size of 6144
AutoRecall      true       ; Migrated HSM files recalled automatically
AutoMount       false      ; Non-mounted volumes not mounted auto.
DirectoryMode   false      ; Use all qualifiers (Datasetmode)
Volume          WTLTCP     ; Volume serial number for new data sets
SpaceType       TRACK      ; Data sets allocated in tracks
Recfm           VB         ; Variable Blocked record format
Filetype        SEQ        ; File Type = SEQ (default)
Smf             170        ; The SMF record sub type to be used
Mgmtclass       TCPMGMT    ; SMS management class for new data sets
Ctrlconn        IBM-850    ; ASCII code page for control connection
Sbdataconn      (IBM-1047,IBM-850) ; EBCDIC, ASCII code page data conn.
Replylanguage   (English,C)    ; English replies
Replylanguage   (Dutch,NL_NL)  ; Dutch replies (does NOT work in V3R1)
Replylanguage   (Danish,DK_DK) ; Danish replies (does NOT work in V3R1)
Quotesoverride  TRUE       ; Single quotes means qualified name
Umask           027        ; Make new HFS files rw- r-- ---
Filetype        SQL   1    ; File Type = SQL
DB2             DSNI  2    ; DB2 subsystem DSNI on mvs18
Spread          False      ; Do not use spread-sheet format
SQLCol          Any        ; Use labels or names
Anonymous FTPANOM/NOSECRET ; The anonymous user ID and password 3
```

1 This server processes all requests as SQL queries by default.

2 The default DB2 subsystem to connect to is called DSNI.

Please note that the OE FTPD server is much less restrictive with respect to processing SQL requests than the non-OE FTPD server is. As each FTP client runs in his or her own MVS address space, two clients can concurrently be connected to two different DB2 subsystems.

3 This server instance allows anonymous login. An anonymous user will be associated with the FTPANOM user ID. The anonymous user will not be prompted for a password.

Assume that the file /u/ftpanom/sqldept.txt has the following contents:

```
SELECT * FROM ALFREDC.ABCDEPT
```

An anonymous user that retrieves this file via this FTPD server can use the following FTP dialogue:

```
[C:\]ftp mvs18o 2021
IBM TCP/IP for OS/2 - FTP Client ver 15:51:28 on Nov 19 1994
Connected to mvs18o.itso.ral.ibm.com.
22O-FTPDSQL1 IBM MVS V3R1 at mvs18oe.itso.ral.ibm.com, 21:06:07 on 1996-
22O Connection will close if idle for more than 5 minutes.
Name (mvs18o): anonymous
23O 'ANONYMOUS' logged on.  Working directory is "/u/ftpanom".
ftp> get sqldept.txt sqldept.out
2OO Port request OK.
125 Sending data set /u/ftpanom/sqldept.txt
25O Transfer completed successfully.
local: sqldept.out remote: sqldept.txt
149O bytes received in 0.44 seconds (3 Kbytes/s)
ftp>
```

The output file, sqldept.out, will hold the result of the SQL query:

```
s--------+---------+---------+---------+---------+---------+---------+--
 SELECT * FROM ALFREDC.ABCDEPT
h--------+---------+---------+---------+---------+---------+---------+--
DEPTNO DEPTNAME                                MGRNO  ADMRDEPT LOCATION
d--------+---------+---------+---------+---------+---------+---------+--
A00    SPIFFY COMPUTER SERVICE DIV.            000010 A00
B01    PLANNING                                000020 A00
C01    INFORMATION CENTER                      000030 A00
D01    DEVELOPMENT CENTER                      ------ A00
D11    MANUFACTURING SYSTEMS                   000060 D01
D21    ADMINISTRATION SYSTEMS                  000070 D01
E01    SUPPORT SERVICES                        000050 A00
E11    OPERATIONS                              000090 E01
E21    SOFTWARE SUPPORT                        000100 E01
F22    BRANCH OFFICE F2                        ------ E01
G22    BRANCH OFFICE G2                        ------ E01
H22    BRANCH OFFICE H2                        ------ E01
I22    BRANCH OFFICE I2                        ------ E01
J22    BRANCH OFFICE J2                        ------ E01
```

The FTP request could also be imbedded in an HTML document, as the following example shows:

```
<A HREF="ftp://mvs18o/sqldept.txt">
Sample DB2 SQL query - <EM>Is now working - check it out</EM></A>
```

By clicking on this link, the Web browser will log on as the anonymous user to the FTP server, send a RETR FTP command for the specified file, sqldept.txt in the default home directory (/u/ftpanom) and display the output from the SQL query. The reason we use the txt suffix, is that the Web browser then will treat the output as a text file and show the result with its default text file browser (see Figure 106).

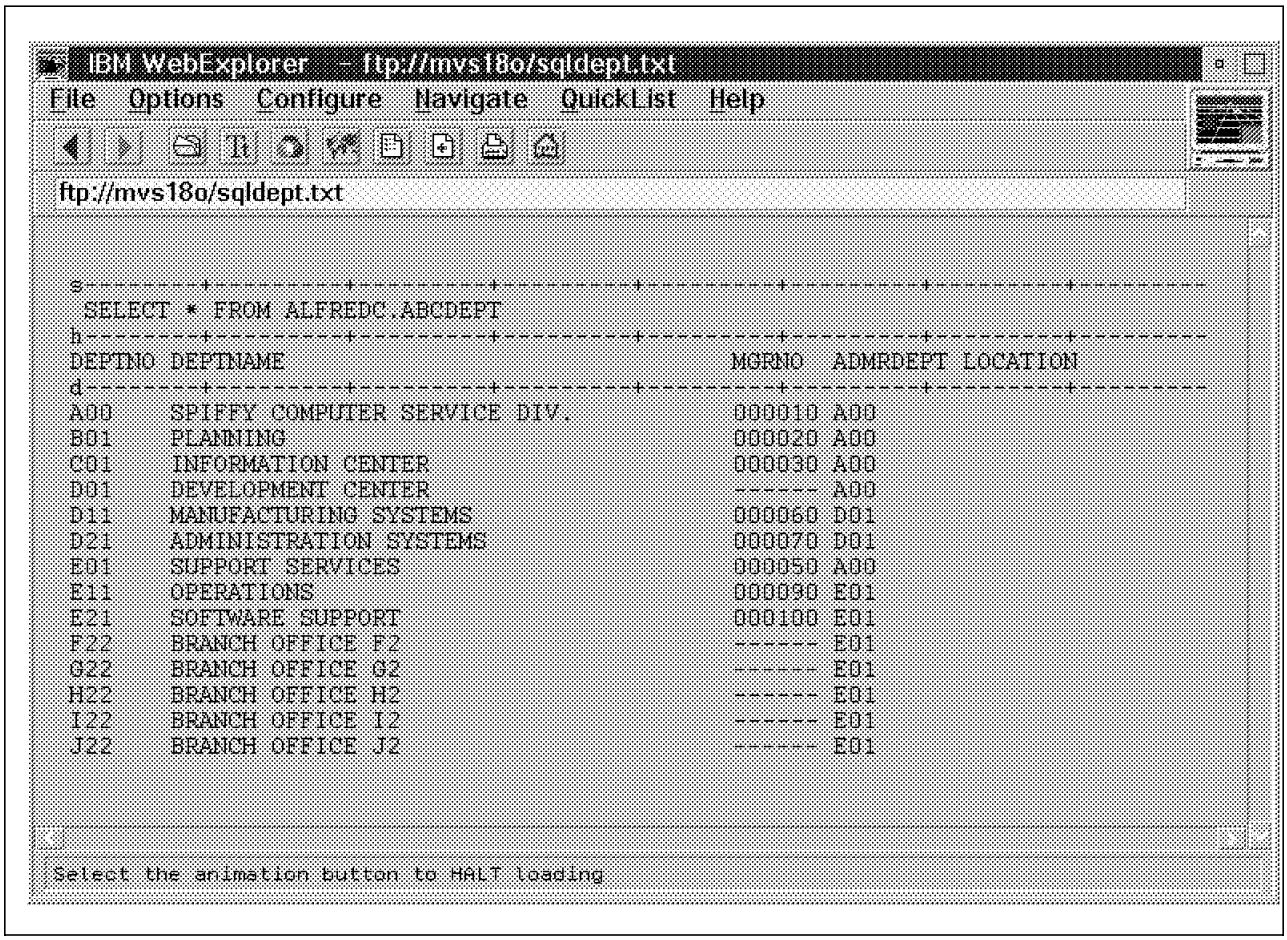


Figure 106. FTP SQL Query from a Web Browser

Note: Please note that this technique will only work if the FTPD server uses the default FTP server port number. Even though the HTTP protocol allows you to specify an alternate port number in an FTP URL, our testing shows that this does not work from the majority of currently available Web browsers.

5.8 ONC/RPC Programming

One of the new functions which the TCP/IP for MVS OpenEdition Applications Feature provides is the integration of the ONC/RPC programming library into the OpenEdition environment. We thought it would be interesting to take an example that was developed on a different platform and see how it can be ported to the OpenEdition UNIX platform. Therefore we fetched a further example from the O'Reilly ftp server (ftp://ftp.ora.com/pub/examples/). Here we looked for the code which is provided with *J. Blohmer: Power Programming with RPC*, published by O'Reilly & Associates. We took as an example the simple remote directory listing (rls). It shows the contents of a directory on a remote host. It consists of six source files and is managed by a makefile.

We had to apply only the following changes:

- We added the following define statements to the source files and could then compile immediately:

  ```
  #ifdef MVS
  /* #define _OE_SOCKETS
  #define _XOPEN_SOURCE_EXTENDED 1 */
  #define _ALL_SOURCE
  #endif
  ```

  Note: The ONC/RPC examples could be compiled with _OE_SOCKETS *and* _XOPEN_SOURCE_EXTEND 1 set as well as with _ALL_SOURCE defined.

- We used a further definition of MVS to isolate our additions. This may be useful to do as it allows you to keep the source current over platforms where it is needed. The conditional compile picks exactly that path of code which is enclosed in the #ifdef/#endif sequence.

- The makefile had to be changed in order to:

  1. Set the compiler variable MVS (**1** in Figure 107) as described above.
  2. Add rules where the default rules of make were not sufficient to compile. For example, we had to add a rule to compile rls_xdr (see Figure 107, **2**) correctly.
  3. A further, minor change had to be done as one of the C routines is defined by POSIX.1 with a different type.

```
cflags = -D MVS  1  -c

ALL = lls rls rls_svc tcpRls

all: $(ALL)

rls_xdr.o: rls_xdr.c rls.h    2
 (CC) -D MVS -c rls_xdr.c

read_dir.o: read_dir.c rls.h
 (CC) -D MVS -c read_dir.c

lls: lls.c read_dir.o
 (CC) -D MVS -g -o lls lls.c read_dir.o

rls: rls.c rls_xdr.o
 (CC) -D MVS -g -o rls rls.c rls_xdr.o -lrpclib

rls_svc: rls_svc.c rls_xdr.o  read_dir.o
 (CC) -D MVS -g -o rls_svc rls_svc.c rls_xdr.o read_dir.o -lrpclib

tcpRls: tcpRls.c rls_xdr.o
 (CC) -D MVS -g -o tcpRls tcpRls.c rls_xdr.o -lrpclib

clean:
 (RM) $(ALL) *.o
```

Figure 107. Makefile to Create a Sample ONC/RPC Application

After we had applied these changes the make ran immediately and we could use the application. We tested it within the OpenEdition environment only, but it should run as a really distributed version as well.

```
pwd=/u/hdm/rpc/rls/manual/minimum: >rls mvs18o /u
.
..
alfredc
cs2201
ftpanom
hdm
juri
kitty
losert
reiko
scadden
suzuki
tcpipweb

pwd=/u/hdm/rpc/rls/manual/minimum: >
```

Figure 108. Output From ONC/RPC Remote Directory Listing

A simple output from the remote directory service is shown in Figure 108.

5.8.1 ONC/RPC OE Port Mapper

The ONC/RPC port mapper is supplied with the TCP/IP for MVS OpenEdition Applications Feature as an OpenEdition application program.

To support ONC/RPC application programs that execute in the OpenEdition environment, you may either start the OE version of the port mapper, or you may start the non-OE port mapper from TCP/IP for MVS as your ONC/RPC port mapper. If you start the non-OE port mapper, you must start it so it runs as a non-OE socket application on the TCP/IP for MVS stack that you use as your OpenEdition AF_INET transport provider.

Your port reservations in the TCP/IP for MVS PROFILE data set must specify OMVS, if you start the OpenEdition version of the port mapper:

```
PORT
   111 UDP OMVS                 ; OE Portmapper Server
   111 TCP OMVS                 ; OE Portmapper Server
```

To start the port mapper as an OpenEdition socket application, you may either start it from the shell or as a started task in MVS.

We chose the started task method to start the port mapper:

```
//PORTMAP PROC
//*
//*  OpenEdition MVS Portmapper Server main process
//*  Resulting address space name will be PORTMAP1, when
//*  we use this method to start the portmapper
//*
//*  OPORTMAP is in SYS1.TCPIP.SEZALINK (on the LINKLST)
//*
//PORTMAP EXEC PGM=OPORTMAP,REGION=4OM,
//        PARM='POSIX(ON),ALL31(ON)/'
```

```
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//         PEND
```

5.8.2 ONC/RPC Archive Libraries

The ONC/RPC routines are supplied as individual object deck files in the /usr/lpp/tcpip/rpc/lib directory. This is where maintenance is applied, but this is not the place where your programs pick them up when you compile and link an ONC/RPC application. You pick them up from a library archive called librpclib.a, which is built from the individual files via an archive command:

```
cd /usr/lpp/tcpip/rpc/lib
ar -qc  /usr/lib/librpclib.a  *.o
```

When you compile and link an ONC/RPC application program, you specify an -l flag:

```
 c89 various parameters -l rpclib
```

The name passed on the -l flag is used to search for an archive file called librpclib.a.

Whenever you apply maintenance to the ONC/RPC routines, you need to manually rebuild the librpclib.a library archive in your /usr/lib directory. A sample shell script is supplied with the TCP/IP for MVS OpenEdition Applications Feature for this purpose; it is located in the /samples directory and it is called tcparrpc. This shell script starts by copying the existing library archive to /usr/lib/librpclib_old.a. If the archive command fails, you need to manually rename this saved copy of your archive library back to its original name before you try to compile and link any ONC/RPC application programs.

5.9 X-Windows Programming

The TCP/IP for MVS OpenEdition Applications Feature brings X-Windows support level X11R6 to the OpenEdition environment. To do a simple installation verification test, several examples are provided. They are distributed in several subdirectories that are located under the /usr/lpp/tcpip/X11R6/Xamples/demos/ directory. We used a PMX 2.0.3 on WARP Connect with CSD UN68122 installed.

Before starting any X-Windows applications, you need to set your DISPLAY environment variable to point to your X-Windows server host:

```
DISPLAY=9.24.104.208:0
export DISPLAY
```

In our sample setup, we included the above two lines in the user's .setup shell script.

Figure 109. X Windows Output from XSAMP3

You should observe the following output from xsamp3 in your OpenEdition session.

```
pwd=/u/hdm/projects/X/xsamp3: >xsamp3
xsamp3 entered.
XtToolkitInitialize done.
XtCreateApplicationContext done.
Display opened.
XmNwidth set.
XmNheight set.
XmNallowShellResize set.
XtAppCreateShell call done.
XtRealizeWidget call done.
XmCreatePushButton call done.
XtManageChild call done.
XtAddCallback call done.
```

Figure 110. Shell Output When Running XSAMP3

5.9.1 X-Windows Archive Libraries

The X-Windows component uses several library archives and you may have to rebuild one or more of these library archives when you apply maintenance to the X-Windows component. If this is the case, the PTF will include a ++HOLD ACT entry that will identify which commands or shell scripts you need to execute after having run your SMP/E job.

Chapter 6. Web Services

The installation and configuration of the Internet Connection Server for MVS has been described and documented in a series of books that accompany the OS/390 Internet BonusPak, which is shipped with OS/390.

The OS/390 Internet BonusPak includes four redbooks that describe how to install and customize the Internet Connection Server for MVS:

- *OS/390 Internet BonusPak: Getting Started*
- *OS/390 Internet BonusPak: Page Management*
- *OS/390 Internet BonusPak: Networking*
- *OS/390 Internet BonusPak: How to Secure the Internet Connection Server for MVS*

The above books can be used in conjunction with the product documentation, *Internet Connection Server for MVS/ESA: User's Guide*, SC31-8204 to plan for and implement the Internet Connection Server for MVS in your MVS/ESA or OS/390 environment.

From an AF_INET transport provider point of view, the Internet Connection Server for MVS is a normal AF_INET socket application that runs in the OpenEdition environment. The issues that are related to configuring the OE resolver and your AF_INET transport provider stacks apply equally well to the Internet Connection Server for MVS as they apply to the TCP/IP for MVS OpenEdition Applications Feature, or, for that matter, to any AF_INET socket application that executes in OpenEdition.

In this chapter, we focus on the following Web-related topics:

- 6.1, Uniform Resource Locators (URL)
- 6.2, Using the OE FTPD Server from Web Browsers
- 6.3, Using the OE TelnetD Server from Web Browsers

© Copyright IBM Corp. 1996

6.1 Uniform Resource Locators (URL)

A URL is the magic glue of the World Wide Web. A URL is a pointer to information somewhere on the Internet. A Web browser uses the URL to fetch the information the end user requests by clicking on a *link* in a Web page. The URL syntax for Internet resources basically consists of three elements:

- The protocol or scheme to use to get the information. Many different protocols are supported. In this context, we focus on the following protocols:

  http Hyper Text Transfer Protocol. This is the protocol that is used by Web servers.

  ftp The information that is referred to by such a URL can be accessed via normal FTP protocols. This is often used if you supply an anonymous FTP service. A Web browser normally starts by logging on as the anonymous user. If this fails, the Web browser may ask the end user for a user ID and password before it tries to access the FTPD server again.

  telnet Establish a telnet connection with the specified internet host.

  Other protocols are supported too, for example, gopher, news, and wais just to mention a few.

- The Internet host identification. This may be a host name or an IP address. The field may optionally include user ID and password information and alternate port number.

- The URL path, which identifies the individual resource on the server host. The exact layout of the URL path field depends on the protocol being used.

URL syntax is described in RFC1738 and RFC1808.

6.2 Using the OE FTPD Server from Web Browsers

If we take a closer look at an FTP URL, the full supported syntax is as follows:

```
ftp://<user>:<password>@<host>:<port>/<path>;type=<typecode>
```

The elements in the above URL are:

user The user ID to use for logging on to the remote FTPD server.

password The password to use when logging on to the remote FTPD server.

Links in an HTML document do not normally include user and password. A Web browser tries to log on as anonymous first; if that fails it requests a user ID and password from the user and uses this in its next attempt to log on to the FTPD server.

You may include only a user ID in your URL. In this case the Web browser will use that user ID and request a password from the end user before it logs on to the FTPD server.

host The Internet host name or IP address.

port An alternate port number.

An FTPD server normally resides on port 21, but if you set up an alternate FTPD server on, for example, port number 2021, you can include this port number in the links you add to your HTML documents.

path The relative or full path of the file to retrieve. Please note that the slash between the host name or port number is *not* part of the path name. To enter a full path name that starts with a slash, you need to use a special encoding of the form %2f. The percent sign means that the two following characters is a hexadecimal encoding of an ASCII character - X′2f′ is the ASCII coding of a slash.

If you enter a relative path name, it is relative to the home directory of the user ID that logs in. If you use anonymous support in the OE FTPD server, it will be the home directory of the anonymous user.

typecode The type code can be used to specify the transfer type to use when the Web browser retrieves the specified file. The Web browser may be able to decide on its own based on the file suffix, if it should use an ASCII or an image transfer type. If you use nonconventional file suffixes, you need to indicate in your HTML links what type of transfer to use. Typical values for the type code are:

a ASCII transfer
i Image transfer (binary)

A third type code of d is documented in RFC1738, but none of the browsers we tested were able to understand such a type code. By using this third type code, a browser should instead of sending a file retrieve command, send a command to retrieve a directory listing from the FTPD server.

Not all Web browsers are smart enough to use the more sophisticated URL formats. So the general recommendation is to keep it simple. Here are a few rules of thumb that should allow you to establish FTP services from a majority of browsers:

1. Enable anonymous log in to your OE FTPD server. Use the form of anonymous log in, where the user is not requested to enter a password. The syntax in your FTP.DATA data set is, for example:

   ```
   Anonymous FTPANOM/NOSECRET ; The anonymous user ID and password
   ```

   Using this setup, your HTML links can have the following simple layout:

   ```
   <A HREF="ftp://mvs18o.itso.ral.ibm.com/%2fu/tcpipWeb/public/ftp1.txt">
   Transfer request for /u/tcpipWeb/public/ftp1.txt, no user ID or
   password, no type code
   </A>
   ```

2. If you do not wish to establish full anonymous log in, we recommend that you include a user ID in your HTML link, but no password. The browsers we worked with were all able to recognize the user ID and prompt the end user for a password before trying to log in to the FTPD server.

   ```
   <A HREF="ftp://userxyz@mvs18o.itso.ral.ibm.com/%2fu/tcpipWeb/public/ftp1.txt">
   Transfer request for /u/tcpipWeb/public/ftp1.txt, user ID=userxyz, no
   password, no type code
   </A>
   ```

   If you configure your FTPD server without anonymous support and do not include a user ID in the HTML links, a Web browser will try to log in as anonymous, and when that fails it should request a user ID and password from the end user and retry the log in. Not all browsers do this. Some just come back and inform the user that the FTPD server does not currently respond to queries and that the user should try again a little later, which of course does not solve the problem.

3. Do not use alternate port numbers for your FTPD server.

   Most Web browsers use what is termed firewall-friendly FTP. Normally an FTP client connects to the FTPD server's control port and when data transfer is required, the server's data port establishes a connection with the FTP client. If the FTPD server is outside a firewall, that means that the firewall must allow TCP connection establishment to take place from outside, which is not considered to be safe. The Web browser can turn this around by sending a PASV FTP command, which turns the server into *passive mode* where the server does not establish the data connection, but waits for the client to do so. By using this technique both control connection and data connection is established from inside the firewall, which makes the IP packet filter rules in the firewall simpler and the implementation generally more secure.

   If you specify an alternate port number for the FTPD server, the data control connection setup works fine, but the data connection setup fails for most of the browsers we used.

6.3 Using the OE TelnetD Server from Web Browsers

You can specify a link that has the following content:

```
<A HREF="telnet://mvs18o.itso.ral.ibm.com">
Sample telnet connection</A>
```

When the end user clicks on such a link, the Web browser will start the local telnet client application and connect to the host specified in the URL. The end user will then have to enter normal user ID and password information in the telnet dialog to gain access to the host.

You can specify both a user ID and a password in the URL, but browsers do not normally pass this information on to the local telnet application, but treat the information as advisory, informing the end user of the suggested user ID and password to type in. The end user still has to type in the information when the local telnet client application has started.

Chapter 7. Tracing OpenEdition Socket Applications

Sooner or later you will get into a situation where you need to analyze why an OpenEdition socket application program does not act as you expected it to act. In this chapter we introduce you to a couple of useful tracing facilities you can use to analyze problems with OpenEdition AF_INET socket applications.

This chapter includes the following topics:

- 7.1, Trace Points
- 7.2, Application Level Trace
- 7.3, Language Environment Library Run-Time Trace
- 7.4, OpenEdition Component Trace
- 7.5, TCP/IP for MVS Socket Trace
- 7.6, TCP/IP for MVS Packet Trace
- 7.7, AnyNet MVS Internal Trace
- 7.8, VTAM Buffer Trace

© Copyright IBM Corp. 1996

7.1 Trace Points

Look at Figure 111 for an overview of the trace points we used a couple of times during the writing of this book.

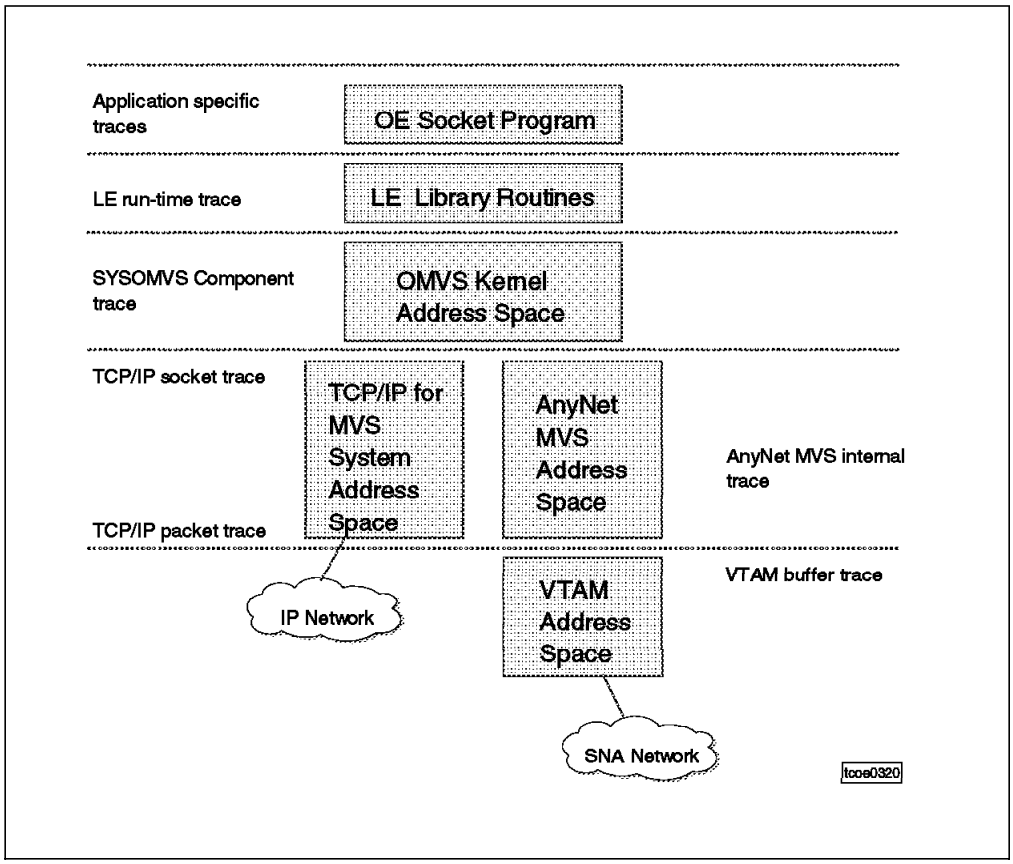


Figure 111. Trace Points

The trace points we describe in this section of the book are the following:

- Application-specific traces

  Most standard socket applications support some kind of application level tracing. If you develop your own socket applications, it will be a good idea to build tracing capabilities into your programs. In many situations an application trace will be sufficient to diagnose a problem in the application design or implementation. We recommend adding time stamps to your own application trace entries, because such time stamps will make it easier to correlate your application trace to some of the other trace facilities that we describe in this chapter (see 7.2, “Application Level Trace” on page 169).

- Language Environment library run-time trace

  This trace will give you a trace entry every time a Language Environment library function is entered and every time a Language Environment library function returns control to the C application program. Return information in terms of return code, errno and errnojr is included on exit trace points. This trace can be very useful if you have C programs that do not check the return code on all library calls, but continue processing even in the case of unexpected return codes. The trace will show you the return codes that are actually passed back to the program (See 7.3, “Language Environment Library Run-Time Trace” on page 174).

- SYSOMVS component trace

  This trace will give you a detailed trace of OE Kernel address space activity. The trace can be started with various options. As a minimum you should run the trace with the SYSCALL option, which will give you trace points every time an OE Kernel address space system call is made and every time the OE Kernel address space returns from a system call. The trace is detailed and requires some insight in OpenEdition processing to analyze (see 7.4, “OpenEdition Component Trace” on page 177).

- TCP/IP for MVS socket trace

  This trace will give you information on all socket calls that are passed from the physical file system to the TCP/IP system address space. The trace includes passed parameters and returned control information. This trace is also very detailed and not always easy to analyze (see 7.5, “TCP/IP for MVS Socket Trace” on page 180).

- TCP/IP for MVS packet trace

  This trace can be used to trace how the IP packets that are exchanged over the network interface actually look. The packet trace can be very useful to determine if the application protocol works as expected. The formatted trace output is relatively easy to understand and analyze. Formatting can be done based on both ASCII and EBCDIC data content (see 7.6, “TCP/IP for MVS Packet Trace” on page 184).

- AnyNet MVS internal trace

  This trace gives you information on the socket calls that are passed from the OpenEdition physical file system to the AnyNet MVS address space along with information on how data is passed between AnyNet MVS and VTAM. Again this is a detailed trace that requires some insight to analyze, but it includes a lot of relevant information (see 7.7, “AnyNet MVS Internal Trace” on page 186).

- VTAM buffer trace

  This trace can show you the exact content of the SNA LU6.2 request units that are exchanged between AnyNet MVS and its partner nodes in your SNA network. A VTAM buffer trace is probably well-known to most SNA installations. The trace data is mostly in ASCII, so it may require some analysis to determine what is going on (see 7.8, “VTAM Buffer Trace” on page 187).

7.2 Application Level Trace

Each application trace is unique in the way it is started and in the way it is captured. We give you an overview of how to start and locate trace information for the standard applications in the OE application feature, and in the Internet Connection Server for MVS.

7.2.1 Telnet Server

The TelnetD server trace is enabled via an option in the InetD configuration file (/etc/inetd.conf):

```
#=================================================================
# service | socket | protocol | wait/ | user | server  | server program
# name    | type   |          | nowait|      | program |   arguments
#=================================================================
#
telnet   stream tcp nowait OMVSKERN /usr/sbin/otelnetd otelnetd -l -m -t
```

The TelnetD server trace is turned on by passing a -t argument to the otelnetd server program. All telnet sessions that are started after this option has been specified will produce trace output.

The trace is written in formatted form to SyslogD facility name local1, which is routed to a file in your Hierarchical File System via a definition in your SyslogD configuration file (/etc/syslogd.conf):

```
#
# All debug messages (and
# above priority messages) from
# telnet go to telnet.debug.a
#
local1.debug              /tmp/syslogd/telnet.debug.a
```

In this example, the trace data is written to /tmp/syslogd/telnet.debug.a in your Hierarchical File System.

See Figure 112 for an example of the TelnetD server trace.

```
                        1
May 16 14:19:31  telnetd[2621446]: EZYTE67I S(nfd):socketfd..ibits=00000000 obits=00000000 ebits=00000000
May 16 14:19:31  telnetd[2621446]: S(nfd) pty..ibits=00000080 obits=00000000 ebits=00000000
May 16 14:19:31  telnetd[2621446]: EZYTE78I PROTOCOL: ptyread 1 chars
May 16 14:19:31  telnetd[2621446]: EZYTE49I Ipt: a3 t   2
May 16 14:19:31  telnetd[2621446]: EZYTU14I UTILITY: netwrite 1 chars.
May 16 14:19:31  telnetd[2621446]: EZYTU21I Ont: 74 .   3
May 16 14:19:32  telnetd[2621446]: EZYTE67I S(nfd):socketfd..ibits=00000001 obits=00000000 ebits=00000000
May 16 14:19:32  telnetd[2621446]: S(nfd) pty..ibits=00000000 obits=00000000 ebits=00000000
May 16 14:19:32  telnetd[2621446]: EZYTE47I Int: 0d0a ..  4
May 16 14:19:32  telnetd[2621446]: EZYTE72I PROTOCOL: netread 2 chars.
May 16 14:19:32  telnetd[2621446]: EZYTU15I UTILITY: ptywrite 1 chars.
May 16 14:19:32  telnetd[2621446]: EZYTU22I Opt: 0d .  5
May 16 14:19:32  telnetd[2621446]: EZYTE67I S(nfd):socketfd..ibits=00000000 obits=00000000 ebits=00000000
May 16 14:19:32  telnetd[2621446]: S(nfd) pty..ibits=00000080 obits=00000000 ebits=00000000
May 16 14:19:32  telnetd[2621446]: EZYTE78I PROTOCOL: ptyread 2 chars
May 16 14:19:32  telnetd[2621446]: EZYTE49I Ipt: 0d15 ..
May 16 14:19:32  telnetd[2621446]: EZYTU14I UTILITY: netwrite 3 chars.
May 16 14:19:32  telnetd[2621446]: EZYTU21I Ont: 0d000a ...
```

Figure 112. Telnet Server Trace

1 This is the process number of the otelnetd process for a single client connection.

2 An EBCDIC character of t comes from the shell process. The byte is translated to its ASCII equivalent (**3**) and sent out over the network to the client.

4 An ASCII carriage-return line-feed sequence is received (X'0D0A'). It is translated to an EBCDIC new line of X'0D' and sent to the shell process (**5**).

The TelnetD server uses the following tokens to represent data flow:

Int Input from the telnet client over the AF_INET socket. Data is in ASCII.

Opt Output from the TelnetD server to the shell process. Data is in EBCDIC.

Ipt Input to the TelnetD server from the shell process. Data is in EBCDIC.

Ont Output to the telnet client over the AF_INET socket. Data is is in ASCII.

So the flow is:

```
Int -> Opt -> Ipt -> Ont
```

See *TCP/IP for OpenEdition MVS: Applications Feature Guide*, SC31-8069 for details on how to interpret the telnet server trace.

7.2.2 Remote Execution and Remote Shell Servers

The traces for both the REXECD server and the RSHD server are enabled via options in the InetD configuration file (/etc/inetd.conf):

```
#======================================================================
# service | socket | protocol | wait/ | user | server  | server program
# name    | type   |          | nowait|      | program |   arguments
#======================================================================
#
shell    stream tcp nowait OMVSKERN /usr/sbin/rshd rshd -d  1
exec     stream tcp nowait OMVSKERN /usr/sbin/rexecd rexecd -d  2
```

The traces are turned on for both servers by passing a -d argument to the server programs. **1** is the RSHD server and **2** is the REXECD server. All commands executed after the debug flags have been turned on in the InetD configuration file and the InetD server has reread the file will produce trace output.

The trace is written in formatted form to the SyslogD facility name daemon with a priority of debug. The trace data is routed to a file in your Hierarchical File System via a definition in your SyslogD configuration file (/etc/syslogd.conf):

```
#
# All ftp, rexecd, rshd
# debug messages (and above
# priority messages) go
# to server.debug.a
#
daemon.debug              /tmp/syslogd/server.debug.a
```

In this example, the trace data is written to /tmp/syslogd/daemon.debug.a in your Hierarchical File System.

See Figure 113 for an example of the REXECD server trace.

```
May 16 17:56:57  rexecd[1638404]: EZYRD03I Remote address = 9.24.104.79
May 16 17:56:57  rexecd[1638404]: EZYRD05I clisecport = 1022
May 16 17:56:57  rexecd[1638404]: EZYRD08I User is: alfredc
May 16 17:56:57  rexecd[1638404]: EZYRD09I Command is: ls -al
May 16 17:56:57  rexecd[1638404]: EZYRD12I Name is: ALFREDC, user is alfredc
May 16 17:56:57  rexecd[1638404]: EZYRD13I dir is: /u/alfredc
May 16 17:56:57  rexecd[1638404]: EZYRD14I uid is: 0, gid is 1
```

Figure 113. REXECD Server Trace

See *TCP/IP for OpenEdition MVS: Applications Feature Guide*, SC31-8069 for details of how to interpret the REXECD and RSHD server traces.

7.2.3 FTP Server

The trace for the FTP server is enabled via options in the FTP server configuration file or via modify commands to the FTP listener process.

You can enable tracing via a TRACE statement in the FTP server configuration file:

```
TRACE
```

There are no parameters on this statement. Tracing will be done for all events for all FTP sessions that are established after the FTP server listener process is started.

If you do not want tracing to be active all the time, you can start your FTP server listener process without a TRACE statement in the FTP configuration file, and then at a later time enable tracing via an MVS modify command to the FTP server listener process. If you start your FTP listener process via a PROCLIB member with the name FTPD, the process to modify is FTPD1.

You have the option of enabling global tracing for all client users, or to enable tracing for a single user. To enable global tracing, you use the following modify command:

```
F FTPD1,TRACE
```

To enable tracing for a single user, you use the following modify command:

```
F FTPD1,UTRACE=userid
```

When the specified user established an FTP connection, tracing will be performed for this user, but not for other users of the FTP server.

Already established FTP connections are not affected by a modify command. Only FTP connections that are established after the modify command was issued will be subject to tracing.

You stop global tracing via the following modify command:

```
F FTPD1,NOTRACE
```

You stop tracing for a single user via the following modify command:

```
F FTPD1,NOUTRACE
```

Already established FTP connections that were started with tracing enabled will continue to produce trace output until the connections are terminated, but new connections will start up without tracing enabled.

The trace is written in formatted form to SyslogD facility name daemon with a priority of debug. The trace data is routed to a file in your Hierarchical File System via a definition in your SyslogD configuration file (/etc/syslogd.conf):

```
#
# All ftp, rexecd, rshd
# debug messages (and above
# priority messages) go
# to server.debug.a
#
daemon.debug              /tmp/syslogd/server.debug.a
```

In this example, the trace data is written to /tmp/syslogd/daemon.debug.a in your Hierarchical File System.

See Figure 114 for an example of the FTP server trace.

```
May 16 18:39:00  ftpd[2752518]: EZY2702I Server-FTP: Initialization completed at 18:39:00 on 05/16/96.
May 16 18:39:00  ftpd[2752518]: EZYFT41I Server-FTP: process id 2752518, server job name FTPD1
May 16 18:39:00  ftpd[2752518]: SK0286 accept_client: calling selectex for socket 6
May 16 18:39:49  ftpd[2752518]: SK0321 accepted client on socket 7
May 16 18:39:49  ftpd[2752518]: SK0349 calling user exit ftchkip with 0918684F 1024 0918687E 21
May 16 18:39:49  ftpd[2752518]: SK0370 New session for 9.24.104.79 port 1024
May 16 18:39:50  ftpd[2752518]: SK0286 accept_client: calling selectex for socket 6
May 16 18:39:51  ftpd[1310723]: SK1244 spawn_ftps: my pid is 1310723 and my parent's is 2752518
May 16 18:39:51  ftpd[1310723]: SK1499 set_new_pgm: issuing execv
May 16 18:39:52  ftps[1310723]: RX: no LANG for child
May 16 18:39:52  ftps[1310723]: RX: child's codeset is: IBM-1047
May 16 18:39:52  ftps[1310723]: RX: child's locale is: C
May 16 18:39:52  ftps[1310723]: RX0672 RX: msgcat is; 88191584
May 16 18:39:52  ftps[1310723]: RX0679 RX: replycat is; 88191664
May 16 18:39:52  ftps[1310723]: SK1511 entering setup_client_sn with socket 7
May 16 18:39:52  ftps[1310723]: RX0324 ftp server client processing entered for socket 7
May 16 18:39:52  ftps[1310723]: RX0333 chkcmdexit successfully loaded
May 16 18:39:52  ftps[1310723]: RX0339 chkpwdexit successfully loaded
May 16 18:39:52  ftps[1310723]: RX0345 chkjesexit successfully loaded
```

Figure 114. FTP Server Trace

Messages from ftpd come from the FTP listener process, while messages from ftps come from a connection-specific FTP server process.

See *TCP/IP for OpenEdition MVS: Applications Feature Guide*, SC31-8069 for details of how to interpret the FTP server trace.

7.2.4 Internet Connection Server for MVS

The trace for the Internet Connection Server for MVS is enabled via options in the JCL you use to start the server.

```
//TCPW3SRV PROC
//WEBSRV   EXEC PGM=IMWHTTPD,REGION=OK,TIME=NOLIMIT,
//   PARM='POSIX(ON),ALL31(ON)/-vv -r /u/tcpipweb/etc/httpd.conf'
//*
//STEPLIB  DD DSN=IMW.V1R1M0.SIMWMOD1,DISP=SHR
//*
//SYSIN    DD DUMMY
//SYSPRINT DD SYSOUT=*
//SYSERR   DD SYSOUT=*
//STDOUT   DD SYSOUT=*
//STDERR   DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//CEEDUMP  DD SYSOUT=*
```

Tracing is enabled via the verbose option:

-v Verbose
-vv Very Verbose
-vc Trace cache operations

Trace data is written to the STDERR DD-name. In the above JCL example, STDERR goes directly to a SYSOUT file.

Tracing applies to all connections to the Internet Connection Server for MVS. The only way to turn tracing off is to restart the server without the verbose options.

Look at Figure 115 on page 174 for an example of the tracing information that is produced by the Internet Connection Server for MVS.

```
          1
05631B8000000000 : server_loop. Waiting for connection...
05633E2000000002 : Timers..... enabled
0563357800000001 : Disabling... gc  caching not enabled either  #1
0563357800000001 : Gc.......... disabled altogether
05631B8000000000 : server_loop. Accepted socket: 4.
05631B8000000000 : server_loop. Attempting thread create
05631B8000000000 : server_loop. Waiting for connection...
056346C800000003 : Thread 1 started at Thu May 16 15:36:42 1996
 Concurrent threads=1.
056346C800000003 : HTHandle.... Socket: 4, LTHD: 5E5EC38, STHD: 5E5F7D0
056346C800000003 : Client sez.. GET / HTTP/1.0
056346C800000003 : Protocol version.... 1.0
```

Figure 115. Internet Connection Server for MVS Trace

1 The prefix on each line is the thread identification.

See *Internet Connection Server for MVS/ESA: User's Guide*, and SC31-8204 for details of how to interpret the Internet Connection Server for MVS trace.

7.3 Language Environment Library Run-Time Trace

This trace is enabled via a run-time option that is passed to the Language Environment run-time environment. The Language Environment run-time options can be altered for a selected application program in more ways:

- Creating an application-specific run-time options module.

  This can be accomplished using the CEEWUOPT sample job in your Language Environment SCEESAMP library. The CEEUOPT module that is created by this sample job must be included when you link-edit your application program, and the run-time options in this module will be in effect for this particular application program.

- Passing Language Environment run-time options to an application program when executing the program.

  If you start your program via MVS JCL, the syntax is:

  ```
  //MYPGM   EXEC PGM=MYPGM,REGION=4OM,
  //        PARM='POSIX(ON),ALL31(ON),TRACE(ON,32K,DUMP,LE=1)/MYPARM'
  ```

  Everything up until the slash ('/') is considered Language Environment run-time options that are passed to the run-time environment. If you need to pass parameters to your application program, they must be placed after the slash.

  If your program starts new processes via fork() or spawn() functions, the run-time options will be passed along to these processes.

- If you start your program from the shell environment, you can pass run-time options to your program via the _CEE_RUNOPTS environment variable. If you use BPXBATCH, it can be done via the STDENV DD statement:

  ```
  //MYPGM   EXEC PGM=BPXBATCH,REGION=4OM,
  //         PARM='pgm /bin/mypgm myparm'
  //STDENV   DD *
  _CEE_RUNOPTS=TRACE(ON,32K,DUMP,LE=1)
  ```

If you start your program from a shell script, it can be done via environment variable assignments. Remember to export the environment variable before you start your program:

```
export _CEE_RUNOPTS='TRACE(ON,32K,DUMP,LE=1) ALL31(ON)'
export _BPX_JOBNAME='MYJOB'
/bin/mypgm myparm &
```

If you start your program via standard MVS JCL, the trace output is written to a DD-name of CEEDUMP. When you look into SDSF to find your trace output, please remember to look for output from your jobname and your jobname suffixed with a letter. If your jobname is MYJOB, output may show up under MYJOB, MYJOB1, MYJOB2, etc., depending on the number of processes that were started as a result of submitting your job.

```
SDSF OUTPUT ALL CLASSES ALL FORMS     LINES 992          DATA SET DISPLAYED
COMMAND INPUT ===>                                        SCROLL ===> CSR
NP   JOBNAME  JOBID    OWNER    PRTY C FORMS    DEST                TOT-REC
 1   MYJOB    STC04097 OMVSKERN    9 Z STD      LOCAL                    95
 2   MYJOB1   A0000637 OMVSKERN    9 A STD      LOCAL                   407
```

1 is the job we submitted. This output file is the CEEDUMP output from the main process.

2 is a forked process. The output file is the CEEDUMP output from the forked process.

If you start your program from the shell environment, the output file is not written to a DD-statement of CEEDUMP, but is written into the Hierarchical File System in the home directory of the user ID under which the process is executing. If the home directory of the current user has not been defined or if it has been set to the root directory, the output file is written to the /tmp directory instead. The name of the file in the Hierarchical File System is /path/CEEDUMP.Date.Time.PID, for example:

```
/u/alfredc/CEEDUMP.19960510.152056.3342339
```

If you cannot find your trace output in SDSF, determine the user ID under which your program executes, look up that user's home directory, and look into that directory for any file name that begins with CEEDUMP.

If you trace standard servers, they will most likely execute under a superuser user ID (for example, OMVSKERN). If OMVSKERN has not been assigned a home directory, look for the trace files in the /tmp directory.

See Figure 116 on page 176 for an extract of a Language Environment trace file.

```
CEE3DMP V1 R5.0: Normal termination occurred while Trace was active.          05/02/96 3:19:28 PM                Page:   1

Information for enclave main

  Information for thread 05645BC800000000

  Language Environment Trace Table:

    Most recent trace entry is at displacement: 002000

       Displacement                      Trace Entry in Hexadecimal                                  Trace Entry in EBCDIC
       ------------ ------------------------------------------------------------------------   --------------------------------
         +000000    Time...... ACCE9A0F89413402    Thread ID......... 05644A7800000000
         +000010    Member ID.... 03    Flags..... 000000    Entry Type..... 00000003
         +000018    94818995 40404040 40404040 40404040  40404040 40404040 40404040 40404040 |main                           |
         +000038    60606E4D F0F3F15D 40869699 924D5D40  40404040 40404040 40404040 40404040 |-->(031) fork()                |
         +000058    40404040 40404040 40404040 40404040  40404040 40404040 40000000 00000000 |                           .....|
         +000078    00000000 00000000                                                       |.....                          |

         +000080    Time...... ACCE9A0FCD6DF802    Thread ID......... 05645BC800000000
         +000090    Member ID.... 03    Flags..... 000000    Entry Type..... 00000004                          1
         +000098    4C606046D F0F3F15D 40D9F1F5 7EF0F0F0  F0F0F0F0 F040C5D9 D9D5D67E F0F0F0F0 |<--(031) R15=00000000 ERRNO=0000|
         +0000B8    F0F0F0F0 40C5D9D9 D5D6F27E F0F0F0F0  F0F0F0F0 00000000 00000000 00000000 |0000 ERRNO2=00000000............|
         +0000D8    00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 |................................|
         +0000F8    00000000 00000000                                                       |.....                          |

         +000100    Time...... ACCE9A0FCD8F7402    Thread ID......... 05645BC800000000
         +000110    Member ID.... 03    Flags..... 000000    Entry Type..... 00000003
         +000118    94818995 40404040 40404040 40404040  40404040 40404040 40404040 40404040 |main                           |
         +000138    60606E4D F0F1F35D 40839396 A2854D5D  40404040 40404040 40404040 40404040 |-->(013) close()               |
         +000158    40404040 40404040 40404040 40404040  40404040 40404040 40000000 00000000 |                           .....|
         +000178    00000000 00000000                                                       |.....                          |

         +000180    Time...... ACCE9A0FCDFBA202    Thread ID......... 05645BC800000000
         +000190    Member ID.... 03    Flags..... 000000    Entry Type..... 00000004                          2
         +000198    4C606046D F0F1F35D 40D9F1F5 7EC6C6C6  C6C6C6C6 C640C5D9 D9D5D67E F0F0F0F0 |<--(013) R15=FFFFFFFF ERRNO=0000|
         +0001B8    F0F0F7F1 40C5D9D9 D5D6F27E F0F5F0C4  F0F1F1C3 00000000 00000000 00000000 |0071 ERRNO2=050D011C............|
         +0001D8    00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000 |................................|
         +0001F8    00000000 00000000                                                       |.....                          |
```

Figure 116. Language Environment Library Run-Time Trace

This example comes from the start of a forked process. From the return code in R15 (**1**), you can determine that this is the child process (RC=0). The parent process receives the process ID of the child on return from the fork() call.

2 is an example of a failing close() call that returns an errno of X'71' and an errnojr reason code of X'011C'. If you look into the appendixes of *Assembler Callable Services For OpenEdition MVS*, SC23-3020, you will see that the problem in this case was that the program closed a file descriptor that was not open.

For more information on analyzing the Language Environment library run-time trace, please see *Language Environment Debugging Guide and Run-time Messages*, SC26-4829.

For more information on how to specify the Language Environment run-time options, please see *Language Environment Installation and Customization Guide*, SC26-4817. The only parameter we occasionally found a reason to change was the size of the trace buffer. To specify a 1MB trace buffer you can use the following options:

```
TRACE(ON,1M,DUMP,LE=1)
```

7.4 OpenEdition Component Trace

This trace is a standard MVS component trace.

To prepare for starting the OpenEdition component trace, you need to define a trace member in SYS1.PARMLIB and an external writer JCL procedure in SYS1.PROCLIB.

1. SYS1.PARMLIB trace members

   The initial OpenEdition trace member is identified via the CTRACE keyword in your BPXPRMxx SYS1.PARMLIB member:

```
BPXPRMxx
--------
 . . .
 . . .
CTRACE(CTIBPX00) ----> CTIBPX00
 . . .                 --------
 . . .                 TRACEOPTS
                                 OFF
                                 BUFSIZE(4M)
                                 OPTIONS('SYSCALL')
```

   We use the initial trace option member to set the size of the trace buffer and to set the SYSCALL trace option as our default SYSOMVS trace option. As we do not want the trace overhead during normal operations, the trace is not activated during startup. To activate the trace with either the SYSCALL options or with any of the other tracing options, and to write the trace records to a trace data set instead of keeping them in storage, we created a second trace member, called CTCBPX01:

```
TRACEOPTS
  WTRSTART(TCTRAC)  1
  ON
  WTR(TCTRAC)  1
  OPTIONS('ALL')  2
```

   1 The external writer called TCTRAC is to be started when this trace member is activated, and trace data is to be handed over to this external writer.

   2 All trace options are to be enabled. An alternative is just to enable, for example, the SYSCALL trace option, which will give you a trace entry for every entry and exit to the OE Kernel address space.

2. SYS1.PROCLIB external writer procedure

   Our sample external writer procedure is called TCTRAC. This procedure must be placed into SYS1.PROCLIB.

```
//TCTRAC PROC
//*
//* External writer for CTRACE
//*
//IEFPROC EXEC PGM=ITTTRCWR
//TRCOUT01  DD DSN=TCPIP.CTRACE1,DISP=OLD,  1
//             UNIT=3390,VOL=SER=WTLTCP,
//             DSORG=PS
//TRCOUT02  DD DSN=TCPIP.CTRACE2,DISP=OLD,  1
//             UNIT=3390,VOL=SER=WTLTCP,
//             DSORG=PS
```

1 We use two trace data sets to be able to capture high-peak activity.

You control the component trace via MVS operator commands:

1. To start the trace issue the following MVS command:

   ```
   trace ct,on,comp=sysomvs,parm=ctcbpx01

   ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND WERE
   SUCCESSFULLY EXECUTED.
   IEE839I ST=(ON,0064K,00064K) AS=ON  BR=OFF EX=ON  MT=(ON,064K) 529
           ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS
    . . .
   AHL906I THE OUTPUT BLOCK SIZE OF    27998 WILL BE USED FOR OUTPUT 557
           DATA SETS:
             TCPIP.CTRACE1
             TCPIP.CTRACE2
   ITT110I INITIALIZATION OF CTRACE WRITER TCTRAC COMPLETE.
   ```

2. To display the status of the trace, issue a display trace command:

   ```
   display trace

   IEE843I 16.40.56  TRACE DISPLAY 600
           SYSTEM STATUS INFORMATION
    ST=(ON,0064K,00064K) AS=ON  BR=OFF EX=ON  MT=(ON,064K)
    COMPONENT MODE  COMPONENT MODE  COMPONENT MODE  COMPONENT MODE
    ------------------------------------------------------------
    IPC       OFF   SYSJES2   OFF   SYSANT00  MIN   SYSSPI    OFF
    SYSJES    OFF   SYSSMS    OFF   SYSOPS    ON    SYSXCF    ON
    SYSLLA    MIN   SYSAPPC   OFF   SYSRSM    OFF   SYSAOM    OFF
    SYSVLF    MIN   SYSOMVS   ON    SYSWLM    MIN
   ```

3. Execute the programs you want to trace. If you want to be able to limit the output when you later format your component trace data sets with IPCS, it may be a good idea to note the address space IDS of the involved address spaces.

4. Stop the trace:

   ```
   trace ct,off,comp=sysomvs

   ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND WERE
   SUCCESSFULLY EXECUTED.
   IEE839I ST=(ON,0064K,00064K) AS=ON  BR=OFF EX=ON  MT=(ON,064K) 691
           ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS

   trace ct,wtrstop=tctrac

   ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT COMMAND WERE
   SUCCESSFULLY EXECUTED.
   IEE839I ST=(ON,0064K,00064K) AS=ON  BR=OFF EX=ON  MT=(ON,064K) 708
           ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS
   AHL904I THE FOLLOWING TRACE DATASETS CONTAIN TRACE DATA : 707
             TCPIP.CTRACE1
             TCPIP.CTRACE2
   ITT120I SOME CTRACE DATA LOST, LAST 1 BUFFER(S) NOT WRITTEN
   ITT111I CTRACE WRITER TCTRAC TERMINATED BECAUSE OF A WTRSTOP REQUEST.
   ```

To format the trace data sets, you have to use IPCS. You can either use IPCS interactively via the IPCS ISPF panels, or you can submit a batch job that formats the trace data sets:

```
//IPCSFMT  JOB 1,IPCS,CLASS=A,MSGCLASS=X
//IEFPROC  EXEC PGM=IKJEFT01,REGION=4M,DYNAMNBR=10
//IPCSPARM DD DSN=SYS1.PARMLIB,DISP=SHR
//IPCSDDIR DD DSN=TCPIP.DDIR,DISP=SHR
//SYSPROC  DD DSN=SYS1.SBLSCLI0,DISP=SHR
//SYSTSPRT DD SYSOUT=*
//IPCSTOC  DD SYSOUT=*
//IPCSPRNT DD SYSOUT=*
//SYSTSIN  DD *
 IPCS
 MERGE  1
 CTRACE DSN('TCPIP.CTRACE1') -
   COMP(SYSOMVS) LOCAL FULL OPTIONS((KERNINFO))  2
 CTRACE DSN('TCPIP.CTRACE2') -
   COMP(SYSOMVS) LOCAL FULL OPTIONS((KERNINFO))
 MERGEEND
 END
/*
```

1 Because we used two trace data sets, we have to use the IPCS MERGE command to merge the contents of the two trace data sets before formatting.

2 For a quick overview of the trace entries, you can specify SHORT instead of FULL. For detailed analysis of the trace, you probably want to format with FULL. The options parameter KERNINFO gives you details on the OE Kernel address space callable services that were used during the trace.

Sample output from the IPCS formatting job is shown in Figure 117 on page 180.

```
                                    SYSCALL...BPX1WRT  MODULE...BPXJCPC 1
02.  SA18      SYSCALL   0F080001  09:05:11.618692  STANDARD SYSCALL ENTRY TRACE 2
    ASID..002F      USERID....ALFREDC
    TCB...006F1618  SYSCALL...00000036
   +0000  00000036  00000000  D1C3E2E2  80000000  | ........JCSS.... |
   +0010  0000000C  849C6050  00000000  00000036  | ....d.-&........ |
   +0020  00000000  00028FF0  00000007  00028F4C  | .......0.......< |
   +0030  00028F50  00028FEC  00028F54  00028FE8  | ...&...........Y |
   +0040  000389E0  80038B50                      | ..i\...&         |
                                    SYSCALL...BPX1WRT  MODULE...BPXFSRDW
02.  SA18      FILE      05700103  09:05:11.618746  CALL TO VN_RDWR
    ASID..002F      USERID....ALFREDC
    TCB...006F1618  SYSCALL...00000036
   +0000  E5E5E540  003800A0  00300190  000000A4  | VVV ...........u |
   +0010  058C1C78  058C1E80  01010024  01000000  | ................ |
   +0020  00000091  C6E4C9D6  00000050  054492B8  | ...jFUIO...&..k. |
   +0030  00000002  00000000  00000000  00000014  | ................ |
   +0040  002FC000  00000000  00000000  00000000  | ..{............. |
   +0050  00000000  00000000  00000000  00000000  | ................ |
   +0060  00000000  00000000  00000000  00000000  | ................ |
   +0070  00000000  2200                          | ......           |
                                    SYSCALL...BPX1WRT  MODULE...BPXFCSRW
02.  SA18      CHARS     05A90504  09:05:11.618836  CHARSPEC CALL TO DEVDRVR WRITE
    ASID..002F      USERID....ALFREDC
    TCB...006F1618  SYSCALL...00000036
   +0000  C3E2E699  89A385C3  00000004  00000000  | CSWriteC........ |
   +0010  22880000  08000000  00000014            | .h.........     |
                                    SYSCALL...BPX1WRT  MODULE...BPXFCSRW
02.  SA18      CHARS     05A90604  09:05:11.618903  RETURN FROM DEVDRVR WRITE
    ASID..002F      USERID....ALFREDC
    TCB...006F1618  SYSCALL...00000036
   +0000  C3E2E699  89A385D9  00000014  00000000  | CSWriteR........ |
   +0010  00000000  228A8000  08000000  00000014  | ................ |
                                    SYSCALL...BPX1WRT  MODULE...BPXFSRDW
02.  SA18      FILE      05700203  09:05:11.618940  RETURN FROM VN_RDWR
    ASID..002F      USERID....ALFREDC
    TCB...006F1618  SYSCALL...00000036
   +0000  D9D9D940  00000014  00000000  00000000  | RRR ............ |
   +0010  C3D1C1D9  00000040  0578D208  00000000  | CJAR... ..K..... |
   +0020  00000000  00002000  00000035  00000000  | ................ |
   +0030  C6E4C9D6  00000050  054492B8  00000002  | FUIO...&..k..... |
   +0040  00000000  00000000  00000014  002FC000  | ..............{. |
   +0050  00000000  00000000  00000000  00000000  | ................ |
   +0060  00000000  00000000  00000000  00000000  | ................ |
   +0070  00000000  00000000  00000000  00000000  | ................ |
                                    SYSCALL...NOOP     MODULE...BPXJCPC
02.  SA18      SYSCALL   0F080002  09:05:11.618975  STANDARD SYSCALL EXIT TRACE 3
```

Figure 117. OpenEdition Component Trace

1 The KERNINFO option on the IPCS CTRACE command gives you syscall name and module name per trace entry.

For each syscall, there is a pair of trace entries: an ENTRY (**2**) and an EXIT (**3**) trace record. If you only enabled the SYSCALL trace option, you would only get these entries. Because the sample output in Figure 117 was created from a trace with the ALL option, more trace entries are included that describe the internal functions of OMVS for the syscall in question.

7.5 TCP/IP for MVS Socket Trace

The TCP/IP for MVS socket trace will give you trace entries per socket call that is passed from the OpenEdition physical file system to the transport protocol layer of TCP/IP for MVS. These socket calls arrive in the TCP/IP for MVS system address space via program calls from the OE Kernel address space.

The trace is activated via a TCP/IP for MVS obeyfile command:

```
TRACE SOCKET
```

The trace is written in formatted form to the SYSDEBUG file of the TCP/IP for MVS system address space (or to the SY1DEBUG, SY2DEBUG, and SY3DEBUG files, if you have defined these in the TCP/IP for MVS system address space).

You stop the trace again via another obeyfile command:

```
NOTRACE SOCKET
```

```
11:43:58 EZB7259I Sock-request called for ACB    103903504:
11:43:58 EZB6710I    IUCV interrupt -> Sock-request (from OE PC Catcher)  1
11:43:58 EZB6718I    Timeout: 8 minutes: 51.155 seconds
11:43:58 EZB6696I    Interrupt type: Pending message
11:43:58 EZB6697I       Path id: 4                                  2
11:43:58 EZB7106I       MsgId 116413656, Length 16, TrgCls: 00190000, Reply len 8, Flags 07
11:43:58 EZB7292I OeReceive: Path PCcatcher 4 (OMVS     BPX00005), msgid 116413656, trgcls 00190000,
                       bfadr1 05671490, bfln1f 16,
11:43:58 EZB5062I 671490:00000002 00000002 00000011 FFFFFFFF
11:43:58 EZB7255I SkSimpleResponse: Client OMVS    , MsgId 116413656, retcode 6 errno 0  3
11:43:58 EZB7294I OeReply: Path PCcatcher 4 (OMVS     BPX00005), msgid 116413656, trgcls 00190000,
                       bfadr2 05671630, bfln2f 8, ret
11:43:58 EZB5062I 671630:00000006 00000000

11:43:58 EZB7259I Sock-request called for ACB    103903504:
11:43:58 EZB6710I    IUCV interrupt -> Sock-request (from OE PC Catcher)
11:43:58 EZB6718I    Timeout: 8 minutes: 51.155 seconds
11:43:58 EZB6696I    Interrupt type: Pending message
11:43:58 EZB6697I       Path id: 4                                  4
11:43:58 EZB7106I       MsgId 116413656, Length 16, TrgCls: 00020006, Reply len 8, Flags 07
11:43:58 EZB7292I OeReceive: Path PCcatcher 4 (OMVS     BPX00005), msgid 116413656, trgcls 00020006,
                       bfadr1 056714B0, bfln1f 16,
11:43:58 EZB5062I 6714B0:0002006F 00000000 000170B8 00000000  5
11:43:58 EZB7255I SkSimpleResponse: Client OMVS    , MsgId 116413656, retcode -1 errno 48  6
11:43:58 EZB7294I OeReply: Path PCcatcher 4 (OMVS     BPX00005), msgid 116413656, trgcls 00020006,
                       bfadr2 05671640, bfln2f 8, ret
11:43:58 EZB5062I 671640:FFFFFFFF 00000030

11:43:58 EZB7259I Sock-request called for ACB    103903504:
11:43:58 EZB6710I    IUCV interrupt -> Sock-request (from OE PC Catcher)
11:43:58 EZB6718I    Timeout: 8 minutes: 51.155 seconds
11:43:58 EZB6696I    Interrupt type: Pending message
11:43:58 EZB6697I       Path id: 4
11:43:58 EZB7106I       MsgId 116413656, Length 0, TrgCls: 00030006, Reply len 8, Flags 87
11:43:58 EZB7107I       PrmMsgHi 0, PrmMsgLo 0
11:43:58 EZB7255I SkSimpleResponse: Client OMVS    , MsgId 116413656, retcode 0 errno 0
11:43:58 EZB7294I OeReply: Path PCcatcher 4 (OMVS     BPX00005), msgid 116413656, trgcls 00030006,
                       bfadr2 05671548, bfln2f 8, ret
11:43:58 EZB5062I 671548:00000000 00000000
```

Figure 118. TCP/IP for MVS Socket Trace

1 You can identify a socket request coming from OpenEdition via this text: (from OE PC Catcher), meaning the call comes in via the OE program call catcher component. Socket requests from OE do not use the IUCV communication path of TCP/IP for MVS. The trace entries may refer to IUCV terminology, but that is only due to internal mapping inside the TCP/IP system address space and has nothing to do with the actual transport mechanism that is used between the OE Kernel address space and the TCP/IP system address space.

2 is a socket() call that returns a socket descriptor number six (**3**).

4 is a bind() call of socket descriptor number six to port number X′6F′ or decimal 111 (**5**). This bind call fails (**6**) with an errno of 48, which means that the address is in use. These calls were issued by the OpenEdition ONC/RPC port mapper that tried to start up and bind its socket to the well-known port number 111. In the TCP/IP for MVS PROFILE.TCPIP data set, we had reserved this port for the non-OE port mapper, which resulted in the above error situation.

See Table 12 on page 183 for an overview of the most important fields in a TCP/IP for MVS socket trace.

Call-type	TrgCls			PrmMsg		Buffer Data in Trace Output
	High Order Bytes		Low Order Bytes	MsgHi	MsgLo	
	Decimal	Hex				
INITAPI	0	0000	0000			Parameters passed on INITAPI call
ACCEPT	1	0001	ssss	0	0	Socket address structure of remote socket
BIND	2	0002	ssss			Socket address structure to bind to
CLOSE	3	0003	ssss	0	0	
CONNECT	4	0004	ssss			Socket address structure to connect to
FCNTL	5	0005	ssss	Cmd	Arg	
GETHOSTID	7	0007	0000	0	0	Host ID of this host (default HOME IP address)
GETHOSTNAME	8	0008	0000	0	0	Host name of this host
GETPEERNAME	9	0009	ssss	0	0	Socket address structure of remote socket
GETSOCKNAME	10	000A	ssss	0	0	Socket address structure of this socket
GETSOCKOPT	11	000B	ssss	level	option name	Value of option
IOCTL	12	000C	ssss			Command and argument
SELECT / SELECTX	13	000D	descrip-tor set size			Select masks
READ / READV	14	000E	ssss	0	0	Received data
RECV / RECVFROM / RECVMSG	16	0010	ssss	0	flags	Received data
LISTEN	19	0013	ssss	0	backlog queue size	
SEND / SENDMSG	20	0014	ssss			Data to be sent
SENDTO	22	0016	ssss	0	0	Flags and data to be sent
SETSOCKOPT	23	0017	ssss	0	0	Option name and value
SHUTDOWN	24	0018	ssss	0	direction	
SOCKET	25	0019	0000			Domain, type and protocol
WRITE / WRITEV	26	001A	ssss	0	0	Data to be written
GETCLIENTID	30	001E	0000	0	0	The client ID of this process
GIVESOCKET	31	001F	ssss			Client ID to give socket to
TAKESOCKET	32	0020	0000			Client ID to take socket from
Note: ssss denotes a socket descriptor number.						

Table 12. Important IUCV Socket Trace Entry Fields

7.6 TCP/IP for MVS Packet Trace

Packet tracing captures IP packets as they enter or leave the device drivers of TCP/IP for MVS. A packet trace shows you the actual IP packets that are exchanged over the IP network. You can analyze IP and TCP or UDP headers as well as your own application data.

The tracing function is implemented in the TCP/IP system address space for those device drivers that are part of the system address space, in the SNALINK LU0 and SNALINK LU6.2 address spaces for the SNALINK devices and finally in the X.25 address space for the X.25 device driver.

You select what you want to trace via the PKTTRACE command, which is passed either to the TCP/IP system address space via an OBEYFILE command or to the other device driver address spaces via an MVS console modify command.

The trace data is collected by MVS Generalized Trace Facility (GTF). You must start a GTF collection address space before you start the actual packet trace function in TCP/IP:

```
//GTFTCPIP PROC MEMBER=GTFPARM
//*
//* TCP/IP GTF procedure
//*
//IEFPROC EXEC PGM=AHLGTF,
//             PARM='MODE=EXT,NOPROMPT,DEBUG=NO,TIME=YES',
//             REGION=2280K,DPRTY=(15,15)
//IEFRDER DD   DSN=TCPIP.V3R1.GTF.TRACE,DISP=SHR
//SYSLIB  DD   DSN=TCPIP.PROCLIB(&MEMBER.),DISP=SHR
```

The IEFRDER DD statement defines the data set that is used to capture the packet trace records.

The GTF parameters for collection of TCP/IP packet trace records are:

```
TRACE=USRP
USR=(5E4)
END
```

In the above example, these parameters are located in TCPIP.PROCLIB(GTFPARM).

TCP/IP uses x'5E4' as GTF event identifier. If you only want to collect the TCP/IP packet trace records in your GTF trace data set, specify the GTF parameters as above.

When GTF has initialized you can start the actual packet trace function in TCP/IP. The following example shows the OBEYFILE data set that we used to start the packet trace:

```
PKTTRACE CLEAR
PKTTRACE PROT=TCP IP=9.24.104.79
TRACE PACKET
```

You can limit the packet trace to certain source or destination ports using a specific protocol on a certain IP address. Please refer to *TCP/IP for MVS Customization and Administration Guide*, SC31-7134, for details about the PKTTRACE command.

You stop the packet trace again via another OBEYFILE command:

```
NOTRACE PACKET
```

After you have stopped the packet trace function in TCP/IP, you can stop the GTF collection address space, and format the GTF trace data set with a TCP/IP utility program called TRCFMT, or you can format the GTF trace data set using your normal IPCS or AMDPRDMP program, which will invoke the TCP/IP formatting routines for the packet trace records.

```
//jobname JOB 1,pgmname,CLASS=A,MSGCLASS=X,NOTIFY=tsouser
//TRACE  EXEC PGM=IKJEFT01
//FMTIN    DD DSN=TCPIP.V3R1.GTF.TRACE,DISP=SHR
//FMTOUT   DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 TRCFMT PRINT=EBCDIC
/*
```

The FMTIN DD statement identifies the trace data set that you specified on the GTF collection address space JCL.

The formatting routines format the protocol headers and dump the user data area in hexadecimal in either EBCDIC or in ASCII translation, depending on your TRCFMT options.

You can request that TRCFMT formats the packet trace not for print but for download to a Sniffer Network Analyzer or a DatagLANce Network Analyzer, if you prefer to use that instead of a printed report.

```
PKT  0000001 DATE=96/05/13 TIME=12:19:40.604325
     FROM LINK=TR1              DEV=LCS_TOKEN_RING
IP   SRC=9.24.104.79        DST=9.24.104.126 1
     VER=4  HDLEN=5  TOS=X'00' TOTLEN=44    ID=316   FLAGS=B'000'
     FRAGOFF=0    TTL=64   PROTOCOL=TCP         CHECKSUM=X'9693'
TCP  SRC=1024        DST=TELNET 2   SEQ=1253260801 ACK=0          HDLEN=6
     WINDOW=28672 CHECKSUM=X'B85D' URGPTR=0     SYN
     OPTION=MAX_SEG_SIZE SIZE=1460

PKT  0000002 DATE=96/05/13 TIME=12:19:40.609589
     TO   LINK=TR1              DEV=LCS_TOKEN_RING
IP   SRC=9.24.104.126       DST=9.24.104.79        VIA=9.24.104.79
     VER=4  HDLEN=5  TOS=X'00' TOTLEN=44    ID=22551 FLAGS=B'000'
     FRAGOFF=0    TTL=60   PROTOCOL=TCP         CHECKSUM=X'43B8'
TCP  SRC=TELNET      DST=1024        SEQ=878223024  ACK=1253260802 HDLEN=6
     WINDOW=28672 CHECKSUM=X'E34F' URGPTR=0     ACK SYN
     OPTION=MAX_SEG_SIZE SIZE=1960

PKT  0000003 DATE=96/05/13 TIME=12:19:40.614593
     FROM LINK=TR1              DEV=LCS_TOKEN_RING
IP   SRC=9.24.104.79        DST=9.24.104.126
     VER=4  HDLEN=5  TOS=X'00' TOTLEN=40    ID=317   FLAGS=B'000'
     FRAGOFF=0    TTL=64   PROTOCOL=TCP         CHECKSUM=X'9696'
TCP  SRC=1024        DST=TELNET      SEQ=1253260802 ACK=878223025  HDLEN=5
     WINDOW=28672 CHECKSUM=X'FD00' URGPTR=0     ACK
```

Figure 119. TCP/IP for MVS Packet Trace

1 This an IP packet coming in over the TR1 link from the IP host with IP address 9.24.104.79.

2 It is a TCP segment coming from an ephemeral port number and going to the telnet server port number (23) on the 9.24.104.126 host, which is the OpenEdition TCP/IP for MVS stack.

The sequence of packets shown in Figure 119 on page 185 is the standard TCP three-way handshake connection setup (SYN, ACK+SYN, ACK). In this situation the TCP connection is between a telnet client and the OpenEdition TelnetD server.

7.7 AnyNet MVS Internal Trace

This trace is produced by the AnyNet MVS address space and includes trace entries for all major activities inside this address space. There are trace entries that relate to socket calls that are passed to AnyNet MVS from the OE Kernel address space, and there are trace entries that relate to the LU6.2 requests that AnyNet MVS makes to VTAM.

The trace is written in formatted form to a DD-statement on the AnyNet MVS address space. You can either use a DD-statement of TRACE to point to a SYSOUT file directly, or you can use a number of TRACEnnn DD-statements, where nnn may range from 1 to 999. You can allocate data sets to these TRACEnnn DD statements and AnyNet MVS will switch between these data sets in a round-robin fashion, filling them up one by one. If no TRACE or TRACEnnn DD statements are defined in the AnyNet MVS start-up procedure, AnyNet MVS does a dynamic allocation of a SYSPRINT DD-statement to a SYSOUT file.

You enable and disable the AnyNet MVS trace function with an AnyNet MVS utility program called ISTSKTRC. To start the trace, you run ISTSKTRC with an *on trace_option* command.

```
//SXTRACE JOB MSGCLASS=O,REGION=4M,CLASS=I
//*
//*  STARTING SXTRACE
//*
//*  TO STOP: OFF ALL
//*
//SXTRACE2 EXEC PGM=ISTSKTRC,
//         PARM=('ON ALL')
//STEPLIB  DD   DSN=SYS1.VTAMLIB,DISP=SHR
//         DD   DSN=SYS1.SISTLMD1,DISP=SHR
//SYSABEND DD   SYSOUT=*
//SYSPRINT DD   SYSOUT=*
```

There are various trace options you can use. The above example enables all internal trace points in AnyNet MVS. See *VTAM AnyNet Feature Guide to Sockets over SNA*, SC31-6559 for the supported trace options.

The sample output shown in Figure 120 on page 187 was created with the ALL trace option.

```
---> getsockname: entered (tcb = 7ed1b0, clni_ptr = 266c88) <--- 1
getsockname: socket = 4
sna_tcp_usrreq(13f00c, PRU_SOCKADDR, 0, 13e500, 0)
getsockname: family = 2, addr = c0a8d201, port = 15, len = 16
---> getsockname: exiting (tcb = 7ed1b0, rc = 0, cc = 0, status = 1) <--- 2
srvr: ipcgetevt returned FFFFFFFD event 0
srvr: ipcgetevt returned 53D9F80 event 53D9E98
srvr: event == IPC_DATA

---> getpeername: entered (tcb = 7ed1b0, clni_ptr = 266c88) <---
getpeername: socket = 4
sna_tcp_usrreq(13f00c, PRU_PEERADDR, 0, 13e500, 0)
getpeername: family = 2, addr = c0a8d209, port = 404, len = 16
---> getpeername: exiting (tcb = 7ed1b0, rc = 0, cc = 0, status = 1) <---
srvr: ipcgetevt returned FFFFFFFD event 0
srvr: ipcgetevt returned 53D9F80 event 53D9E98
srvr: event == IPC_DATA

---> ioctl: entered (tcb = 7ed1b0, clni_ptr = 266c88) <---
ioctl: socket = 4, cmd = 8004a77c
sna_tcp_usrreq(13f00c, PRU_CONTROL, 8004a77c, 53d9ee4, 0)
---> ioctl: exiting (tcb = 7ed1b0, rc = -1, cc = 1112, status = 1) <--- 3
srvr: ipcgetevt returned FFFFFFFD event 0
srvr: ipcgetevt returned 53D9F80 event 53D9E98
srvr: event == IPC_DATA
```

Figure 120. AnyNet MVS Internal Trace

For every socket call that is passed to AnyNet MVS, the trace will contain a call entry trace point (**1**), and a call exit trace point (**2**).

3 If a socket call fails, the exit trace point will include return code and error number. In this example, the return code is minus one, indicating error return, and the error number is 1112, meaning that the operation is not supported on the socket. These codes can be found in Appendix A of *VTAM AnyNet Feature Guide to Sockets over SNA*, SC31-6559. The specific error situation shown above was provoked by our testing. AnyNet MVS does support the ioctl() call with the specified option.

7.8 VTAM Buffer Trace

The VTAM buffer trace captures SNA request units as they pass in and out of the VTAM buffer for the specified logical unit.

The VTAM buffer trace takes place in the VTAM address space and is enabled via a VTAM modify trace command. The trace data is collected by MVS Generalized Trace Facility (GTF). You must start a GTF collection address space before you start the VTAM buffer trace. You can use the same GTF procedure as the one you use for TCP/IP packet trace (see 7.6, “TCP/IP for MVS Packet Trace” on page 184). The GTF parmlib member must specify the following options:

```
TRACE=USRP
USR=(FE1,FE2,FE3,FE4,FEF,FF0,FF1,FF2)
END
```

You need to know the AnyNet MVS LU name in order to start the trace. If the LU name is SNACK000, you enable the trace using the following VTAM modify command:

```
f net,trace,id=snack000,type=buf

IST097I MODIFY ACCEPTED
IST513I TRACE INITIATED FOR NODE SNACK000
```

You stop the trace again with the following VTAM modify command:

```
f net,notrace,id=snack000,type=buf

IST097I MODIFY ACCEPTED
IST512I TRACE TERMINATED FOR NODE = SNACK000
```

There are various ways you can format and print the trace data collected by GTF. We recommend using the ACFTAP utility program.

```
//jobname  JOB 1,pgmname,CLASS=A,MSGCLASS=X,NOTIFY=tsouser
//ACFTAP   EXEC  PGM=ACFTAP,REGION=6M
//SORTLIB  DD DSN=SYS1.SORTLIB,DISP=SHR
//STEPLIB  DD DSN=SSP.V4R1.SSPLIB,DISP=SHR
//SYSTRACE DD DSN=TCPIP.V3R1.GTF.TRACE,DISP=SHR
//SYSIN    DD *
INPUT=ALL
SOURCE=GTF
NODE=
SUMMARY=NO
LINECNT=60
PRINT=YES
DUMP=NO
SDPRT=YES
DTPRT=YES
SSPRT=YES
GO
QUIT
/*
//SORTIN    DD DSN=TAPSORTI.DATA.TAP,UNIT=SYSDA,SPACE=(CYL,(10,5)),
//             DISP=(NEW,DELETE),DCB=(RECFM=F,LRECL=364,BLKSIZE=364)
//SORTOUT   DD DSN=TAPSORTO.DATA.TAP,UNIT=SYSDA,SPACE=(CYL,(10,5)),
//             DISP=(NEW,DELETE),DCB=(RECFM=F,LRECL=364,BLKSIZE=364)
//SORTWK01  DD DSN=&TEMPD5,UNIT=SYSDA,SPACE=(CYL,(10,5),,CONTIG),
//             DISP=(NEW,DELETE)
//SYSTEMP1  DD DSN=TAPTEMP1.DATA.TAP,UNIT=SYSDA,
//             SPACE=(CYL,(10,5)),DISP=(NEW,DELETE),
//             DCB=(RECFM=F,LRECL=269,BLKSIZE=269)
//SYSTEMP2  DD DSN=TAPTEMP2.DATA.TAP,UNIT=SYSDA,
//             SPACE=(CYL,(10,5)),DISP=(NEW,DELETE),
//             DCB=(RECFM=F,LRECL=269,BLKSIZE=269)
//SYSLDPRT  DD SYSOUT=*
//SYSLSPRT  DD SYSOUT=*
//SYSGSPRT  DD SYSOUT=*
//SYSSDPRT  DD SYSOUT=*
//SYSSSPRT  DD SYSOUT=*
//SYSNEPRT  DD SYSOUT=*
//SYSDTPRT  DD SYSOUT=*
//SYSVTPRT  DD SYSOUT=*
//SYSLUPRT  DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//SYSIXPRT  DD SYSOUT=*
//SYSNTPRT  DD SYSOUT=*
//SYSNPPRT  DD SYSOUT=*
//SYSCSPRT  DD SYSOUT=*
```

```
//SYSCAPRT  DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
```

ACFTAP produces various reports based on the buffer trace of the logical unit in question. See Figure 121 for a sample extract of a trace during ping operations from an AnyNet access node to the AnyNet MVS node.

```
0000023 DATA FLOW   40 00 10 10 20 00 00 8D 00 00 00 12 00 00 00 18 1C 00 02 3C 06 79 00 02 00 74 00 95 00 00 71 8F 00 [1]
                    00 1B 2A 02 01 05 C0 A8 D2 01 03 00 00 02 01 05 C0 A8 D2 08 03 00 00 02 0A 45 00 00 54 14 3B 00 00
                    FF 01 82 12 C0 A8 D2 08 C0 A8 D2 01 08 00 A2 64 66 00 01 00 52 7B 97 31 10 EB 09 00 08 09 0A 0B 0C
                    0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D
                    2E 2F 30 31 32 33 34 35 36 37

                    TIMESTAMP: 18.11.17.240752
       TH  00-02    FORMAT ID (FID):  4 * TG SWEP:OFF MIG:OFF PCI:OFF * NET PRI:OFF IERN: 01 ERN: 00 * [2]
       TH  03-04    VR NUMBER (VRN):  1 * VRCWI: INC  TG REORDR REQD: 0 * TP PRI:  0       TG SEQUENCE NUMBER:   000 *
       TH  04-06    VRCWRI: R VRRWI:  0 * SINGLY SEQUENCED DATA         *                 VR SEQUENCE NUMBER:   08D *
       TH  06-      VR PACING: NONE     * ORIGIN: 00000018 0679         *               SNF SEQUENCE NUMBER:  0002 *
       TH    -25    SEGMENT(MPF):ONLY * DESTINATION: 00000012 023C * FLOW: NORMAL          COUNT (DCF): 00116 *

       RH  00-02    RU TYPE: FM DATA FLOW      REQUEST  * RESPONSE/REQUEST: DR1     ERI      * CHAIN: MIDDLE OF CHAIN* [3]
                    RU FORMAT: UNFORMATTED              * PACING INDICATOR: ON               *                        *
                    BRACKET:                            * CHANGE DIRECTION INDICATOR: OFF    * CODE SEL:EBCDIC *

0000023 USER DATA   *...........yK........yK................b..yK..yK...s......#p.....................................* [4]
                    *.................                                                                                  *


0000024 DATA FLOW   40 00 10 10 20 00 00 8E 00 00 00 12 00 00 00 18 1C 00 02 3C 06 79 00 03 00 74 00 90 00 00 71 8F 00
                    00 1B 2A 02 01 05 C0 A8 D2 01 03 00 00 02 01 05 C0 A8 D2 08 03 00 00 02 0B 45 00 00 54 14 3C 00 00
                    FF 01 82 11 C0 A8 D2 08 C0 A8 D2 01 08 00 A0 64 66 00 02 00 53 7B 97 31 10 EB 09 00 08 09 0A 0B 0C
                    0D 0E 0F 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D
                    2E 2F 30 31 32 33 34 35 36 37

                    TIMESTAMP: 18.11.17.241186
       TH  00-02    FORMAT ID (FID):  4 * TG SWEP:OFF MIG:OFF PCI:OFF * NET PRI:OFF IERN: 01 ERN: 00 *
       TH  03-04    VR NUMBER (VRN):  1 * VRCWI: INC  TG REORDR REQD: 0 * TP PRI:  0       TG SEQUENCE NUMBER:   000 *
       TH  04-06    VRCWRI: R VRRWI:  0 * SINGLY SEQUENCED DATA         *                 VR SEQUENCE NUMBER:   08E *
       TH  06-      VR PACING: NONE     * ORIGIN: 00000018 0679         *               SNF SEQUENCE NUMBER:  0003 *
       TH    -25    SEGMENT(MPF):ONLY * DESTINATION: 00000012 023C * FLOW: NORMAL          COUNT (DCF): 00116 *

       RH  00-02    RU TYPE: FM DATA FLOW      REQUEST  * RESPONSE/REQUEST: DR1     ERI      * CHAIN: MIDDLE OF CHAIN*
                    RU FORMAT: UNFORMATTED              * PACING INDICATOR: OFF              *                        *
                    BRACKET:                            * CHANGE DIRECTION INDICATOR: OFF    * CODE SEL:EBCDIC *

0000024 USER DATA   *...........yK........yK................b..yK..yK.........#p.....................................*
                    *.................                                                                                  *
```

Figure 121. VTAM Buffer Trace

1 This is the full unformatted unit of data that is exchanged over the LU6.2 session. The emphasized portion is the IP datagram that is imbedded in this SNA request unit.

The SNA transmission header is formatted in **2**, while the SNA request header information is formatted in **3**.

4 This is an EBCDIC representation of the user data section of the SNA request unit. As most of the data will be in ASCII, you will probably need to look into the unformatted section (**1**) to see the actual contents.

Chapter 8. Putting Things Together

During the work with this book, we encountered a couple of situations where we had to dig into areas that were actually outside the scope of this book. We decided to include these in this chapter even though they are not directly related to the purpose of the book. You might find some of them useful.

This chapter includes the following topics:

- 8.1, Data Exchange and Access
- 8.2, File Compression
- 8.3, Square Bracket Problem
- 8.4, Compiling C Code on OpenEdition
- 8.5, Raw Socket Usage
- 8.6, Time Distribution

© Copyright IBM Corp. 1996

8.1 Data Exchange and Access

The implementation of the Hierarchical File System is transparent to the type of data to be stored. Data may be text or binary; the implementation does not care about this. However, you may need to know what kind of data you are working with when you exchange data with other systems:

- Text Data

  In a client/server environment where MVS participates, you have to consider how to do ASCII-EBCDIC translation. At first this might seem to be only a matter of two translation tables, but it goes much further than that. As soon as you have to deal with different countries, you have to handle translations using the appropriate code page. OpenEdition provides in the shell environment a command (ICONV), and for programs coded in C a conversion routine iconv(). Both allow you to convert according to the code pages that are applicable.

- Binary Data

  Binary data is much more complicated to handle than text data. Integer variables may be stored differently according to the byte order in storage (little endian (PC) versus big endian (S/390)). Different representations of floating point variables exist. There is no single simple solution for handling such differences. Each situation has to be analyzed to find a suitable solution.

In an ONC/RPC application as well as in a DCE/RPC application, the programming interface layer takes care of these translations and conversions.

In an environment where S/390 provides the server platform, you normally can make a trade-off based on what kind of coding is mostly needed. If workstation users store and fetch data in the Hierarchical File System based on NFS or FTP and further processing on the MVS system is limited, then you may decide to store all data in the representation it has on the workstations.

To transfer files in and out of your OpenEdition system, you can use the FTPD server from the TCP/IP for MVS OpenEdition Applications Feature. With this server, you can transfer files directly in and out of the Hierarchical File System, as well as in and out of traditional MVS data sets. We used the FTPD server for transferring both binary data (see below) and text data (C source code). We transferred both single files, tar files, and pax files. The latter are collections of files which make it easier to transfer a bunch of files and even allow you to reconstruct the directory structure. We recommend that you use one of them when exchanging data which span several directories. The drawback is that you have to separate text and binary data.

Note: We did not use cpio type archives, but they are supported as well.

As we mentioned before, file archives of tar or pax format are very convenient to transfer files that are organized in a directory tree. The pax shell command allows you to expand the various kinds of archive file formats. It allows you to translate data from ASCII to EBCDIC or vice versa at the same time. See Figure 122 on page 193 for an example of how to extract a tar type archive and translate from ASCII to EBCDIC at the same time.

```
pax -o from=ISO8859-1,to=IBM-1047 -x tar -rf your_tar_file.tar
```

Figure 122. pax Command to Expand and Translate a tar Archive

Data exchange between traditional MVS data sets and the Hierarchical File System can be accomplished through the TSO/E copy commands that are provided with OpenEdition, OCOPY, OPUT, OGET, OPUTX and OGETX.

- OCOPY is based on DDNAME allocation. So you have to allocate your input and output DDNAMEs to MVS data sets or HFS files before you invoke the OCOPY command.
- OGET, OPUT, OGETX and OPUTX give you the opportunity to specify the MVS data set name and HFS file name directly on the command invocation.

  OGETX and OPUTX support partitioned data sets, allowing you to copy all members of a partitioned data set in one command invocation. In addition you may apply a suffix to the new files. If, for example, you want to copy all members of your partitioned library *hlq.project.c* to the Hierarchical File System as files in a specific directory, you can have all member names changed to file names of the form *membername.c* while you copy.

All commands support ASCII to EBCDIC conversion.

When you copy from a traditional MVS data set that contains binary data you should use the BINARY option of the OPU or OPUTX command (see Figure 123).

```
oput gzip.exe '/u/hdm/gzip' binary
```

Figure 123. OPUT Command with Binary Option

In case you copy a data set that contains text type data in ASCII encoding and you want to convert the data to EBCDIC while copying to the Hierarchical File System, use the command format that is shown in Figure 124.

```
oput love.letter '/u/hdm/love.letter' binary CONVERT((BPXFX111))
```

Figure 124. OPUT Command With Translation from ASCII to EBCDIC

Note: You need to code *both* brackets for the CONVERT keyword parameter.

The BINARY option is important, because it tells OPUT to handle the input data set as a string of bytes, not as a collection of records.

8.2 File Compression

One of the compression programs in the world of UNIX is the GZIP implementation. It is available through various ftp sites on the Internet, for example, the *Hobbes Virtual Mirror* site provided by TeamOS/2 (*http://www.teamos2.org*). From there we fetched Version 1.2.4. We uploaded all C source and header files from the main directory (GZIP-1.2.4) directly into the Hierarchical File System. We used the ASCII-to-EBCDIC conversion provided by the OpenEdition FTPD server. With a simple makefile shown in Figure 125 on page 194, we could produce the executable module. Actually the makefile is

one of the makefiles provided with the package. There were only minor changes necessary to get it working for the make utility in OpenEdition.

```
flags=-I. -c
CCC=c89
#
# Object files
#
OBJS=gzip.o zip.o deflate.o trees.o bits.o unzip.o inflate.o \
 util.o crypt.o lzw.o unlzw.o unpack.o getopt.o unlzh.o
#
# How to build .o's from .c's
#
.c.o:
 $(CCC)  $(cflags) $(cvarsdll) $<

#
# Main target
#
all: gzip.exe

#
# Link target. setargv.o is provided in the compiler library directory.
#
gzip.exe: $(OBJS)
       $(CCC) -o gzip  $(OBJS)

#
# Dependencies
#
gzip.o:        gzip.c gzip.h tailor.h
zip.o:         zip.c gzip.h tailor.h crypt.h
deflate.o:     deflate.c gzip.h tailor.h
trees.o:       trees.c gzip.h tailor.h
bits.o:        bits.c gzip.h tailor.h crypt.h
unzip.o:       unzip.c gzip.h tailor.h crypt.h
inflate.o:     inflate.c gzip.h tailor.h crypt.h
util.o:        util.c gzip.h tailor.h
lzw.o:         lzw.c gzip.h tailor.h
unlzw.o:       unlzw.c gzip.h tailor.h revision.h lzw.h
unpack.o:      unpack.c gzip.h tailor.h
crypt.o:       crypt.c gzip.h tailor.h
unlzh.o:       unlzh.c gzip.h tailor.h lzw.h
getopt.o:      getopt.c getopt.h

clean:
 -rm $(OBJS)

clobber: clean
 -rm gzip.exe
```

Figure 125. Makefile to Create GZIP

After running make GZIP is ready to use as you can see in on Figure 126 on page 195.

```
pwd=/u/hdm/projects/gzip: >gsip -h
gsip 1.2.4 (18 Aug 93)
usage: gsip [-cdfhlLnNtvV19] [-S suffix] [file ...]
 -c --stdout      write on standard output, keep original files unchanged
 -d --decompress  decompress
 -f --force       force overwrite of output file and compress links
 -h --help        give this help
 -l --list        list compressed file contents
 -L --license     display software license
 -n --no-name     do not save or restore the original name and time stamp
 -N --name        save or restore the original name and time stamp
 -q --quiet       suppress all warnings
 -S .suf  --suffix .suf     use suffix .suf on compressed files
 -t --test        test compressed file integrity
 -v --verbose     verbose mode
 -V --version     display version number
 -1 --fast        compress faster
 -9 --best        compress better
 file...          files to (de)compress. If none given, use standard input.
pwd=/u/hdm/projects/gzip: >
```

Figure 126. GZIP -h Verification Call

Note: An unbelievable number of tools and services is available on the Internet and we described the implementation of only one of those into the OpenEdition environment here. But, we emphasize that by doing this we do *not* recommend that you use these programs in your daily business. MVS, with all it major software components and the underlying hardware, provides a mechanism to store data as efficiently as possible. But if you need to handle GZ files, you now know how to do so.

8.3 Square Bracket Problem

When you use OMVS functions from 3270 terminals, you will often encounter problems with the square brackets ([,]). This is true even in a native US environment that uses host code page 037. We found a relatively simple solution to this problem, which we describe briefly here. This solution applies to Communications Manager/2 Version 1.11.

- To find out which host code page you are using, click on **Keyboard** in an active 3270 window. Select **Remap keyboard** from the pull-down menu. The remapping utility starts. Click on **File** on the menu bar and select **Properties**. Now you should see a panel that includes information about the active host code page.
- CM/2 provides a tool to extract all code page tables from its internal database. This tool is called ACSGCCRT.EXE and it creates *.cpt files. Switch to cmlib and run this tool.
- Assuming you are using host code page 037, make a backup copy of 037.cpt.
- Now edit 037.cpt and apply the following changes (see Figure 127 on page 196 for the original version of 037.cpt and Figure 128 on page 196 for the changed version of 037.cpt).

  1. Change the lines that assign Left_Bracket to BA and Right_Bracket to BB to both *Undefined*. Then change the transformation of Y_Acute_Capital

into AD to Left_Bracket. Change transformation of Diaeresis/Umlaut_Accent into BD to Right_Bracket.

- In addition to this, it may be necessary to remap your keyboard so the keys that have the square brackets engraved are really assigned to square bracket characters. Go to the keyboard remap as described above, click on the corresponding keys, and change both if necessary.

```
Y_Acute_Capital                 AD
Thorn_Icelandic_Capital         AE
Registered_Trademark_Symbol     AF
Circumflex_Accent               B0
Pound_Sterling_Sign             B1
Yen_Sign                        B2
Middle_Dot                      B3
Copyright_Symbol                B4
Section_Symbol                  B5
Paragraph_Symbol                B6
One_Quarter                     B7
One_Half                        B8
Three_Quarters                  B9
Left_Bracket                    BA
Right_Bracket                   BB
Overline                        BC
Diaeresis/Umlaut_Accent         BD
```

Figure 127. Original Version of 037 as Provided with CM/2

```
Left_Bracket                    AD
Thorn_Icelandic_Capital         AE
Registered_Trademark_Symbol     AF
Circumflex_Accent               B0
Pound_Sterling_Sign             B1
Yen_Sign                        B2
Middle_Dot                      B3
Copyright_Symbol                B4
Section_Symbol                  B5
Paragraph_Symbol                B6
One_Quarter                     B7
One_Half                        B8
Three_Quarters                  B9
Undefined                       BA
Undefined                       BB
Overline                        BC
Right_Bracket                   BD
```

Figure 128. Modified Version of 037

For other code page tables, the procedure described here may also apply. But additional problems may occur with the other POSIX invariant characters. See Figure 129 on page 197.

```
Right brace              }
Left brace               {
Backslash                \
Right square bracket     ]
Left square bracket      [
Circumflex               ^
Tilde                    ~
Exclamation point        !
Pound sign               #
Vertical bar             |
Dollar sign              $
Commercial at-sign       @
Accent grave             `
```

Figure 129. POSIX Invariant Characters

Assuming your host code page is 273 (German) you will have duplicate assignment of code points. The characters ä and ü match { and } respectively. So a general solution is not that simple.

It should be pointed out here that this is not a general recommendation. A different approach may be to customize an OpenEdition user conversion table, BPXFxxx. But, in this case you have a solution only in the OMVS shell environment, not in the ISPF edit environment.

8.4 Compiling C Code on OpenEdition

C is the prevailing programming language in the UNIX environment. Two compilers exist on MVS/ESA that support the OpenEdition environment:

- 5688-216 IBM SAA AD/Cycle C/370 Version 1
- 5655-121 IBM C/C++ for MVS/ESA Version 3

Both of them support the OpenEdition environment, which means that you can use both to compile XPG 4.2 compliant code. To use the AD/Cycle compiler you will need Release 2.

As the C/C++ compiler is a member of IBM′s family of C Compilers it is recommended to use this compiler. In addition to compiling pure C code, Version 3 Release 2 allows you to use object-oriented C++ in the OpenEdition environment.

8.4.1 Pitfalls

C is a high-level, multi-purpose programming language. It can be used to write commercial applications as well as system programs. Most of the original UNIX kernel has been written in C. This can be seen in many details that C offers its users. And these details make it difficult for a newcomer to use C efficiently and appropriately. Some people say, C is assembler *de luxe*.

We already mentioned that C allows you to create highly portable code. This means your code may compile even without any modification or adaptation. But there are a couple of things that have to be taken care of:

- The C language allows you to code the following:

```
char z;
z = '7' ;
if (47 < z && z < 58) { /* is numeric */
some code here
}
```

What the code does is to check if the *binary* representation of *z* lies in the region between decimal 47 and 58. If you look up an ASCII table you will find that the numbers 0-9 are located at these code points. This code will compile immediately without any error on every machine.

What happens if such a piece of code is ported to the OpenEdition environment and translated to EBCDIC? It will definitely fail. Thoughtful testing is required even if the source code compiles immediately.

C provides services which makes such a piece of code portable even across ASCII and EBCDIC platforms.

A similar problem occurs because the ASCII characters are stored *contiguously* in the code table, where the EBCDIC characters are not. The following do-loop runs nicely and makes sense in an ASCII environment.

```
for (i='A';i<='Z';i++) &lrbc. do something here }
```

As the alphabet in an EBCDIC environment is not stored contiguously this piece of code is not directly portable to OpenEdition.

- Another issue is when your program relies on the byte order of data. This applies mostly to integer data that has been transmitted in binary, for example, data that contains integer or floating point variables. To avoid such problems, data should be translated to text strings, transmitted as such, and finally re-created as binary data. Another alternative is to write your own data exchange program that allows you full control over both transfer and data format.

8.4.2 C Syntax

The C/C++ Compiler for MVS/ESA checks that variables on both sides of an assignment are of the same type. A typical assignment statement like the following is accepted by many compilers:

```
char g[10] ;
g[0] = NULL ; /* Have really a string of length 0 */
```

The C/C++ Compiler for MVS/ESA complains about this with an error message. This situation like other similar assignment checks, can easily be resolved by *type-casting*:

```
g[0] = (char) NULL ; /* Have really a string of length 0 */
```

The casting of a NULL to be a character NULL solves the situation described above.

8.4.3 Header Files

C programs require several definitions that are typically included in header files. If your source code contains string handling routines, you should include strings.h in your source code. Through these include files, all definitions of symbols, macros, functions, etc., are included and made available to the compiler.

Here we discuss the options that control the include behavior as it is required for socket or ONC/RPC applications. The following settings are possible:

- _OE_SOCKETS

  Defines a BSD-like socket interface for the function prototypes and structures involved. This can be used with _XOPEN_SOURCE_EXTENDED 1 and the XPG4.2 socket interfaces will be replaced with the BSD-like interfaces. An application cannot specify _OE_SOCKETS with _OPEN_SOCKETS. A compile time error message will be generated.

- _OPEN_SOCKETS

  Defines a BSD-like socket interface for the OpenEdition Application Services (FMID HOT11x0). This option cannot be used with either _XOPEN_SOURCE_EXTENDED 1 _OE_SOCKETS, or _OPEN_SOURCE=2. A compile time error message will be generated.

  Use of this feature test macro implies that the application is using the OpenEdition Application Services (FMID HOT11x0) product implementation of the sockets run-time Library library, and must comply with several other requirements, such as header concatenation and inclusion of HOT11x0 unique headers.

- _ALL_SOURCE

  Defines all of the functionality that is currently available on OpenEdition MVS, including XPG4, XPG4.2, and all of the OpenEdition MVS extensions. In addition, defining _ALL_SOURCE makes a number of symbols visible that are not permitted under ANSI, POSIX or XPG4, but which are provided as an aid to porting C-language applications to OpenEdition MVS.

Note: A detailed discussion of these macros can be found in *IBM C/C++ for MVS/ESA Library Reference*, SC23-3881.

The ONC/RPC samples could be compiled with _OE_SOCKETS *and* _XOPEN_SOURCE_EXTEND 1 set as well as with _ALL_SOURCE defined (see Appendix D, “Sample ONC/RPC Application” on page 223).

Note: If a source program can be ported to OpenEdition just by defining _ALL_SOURCE, then it is possibly to set this option on the command line invocation of the compiler:

```
c89 -D _ALL_SOURCE ....
```

The possible settings are shown in Table 13.

Table 13 (Page 1 of 2). Feature Test Macros and Standards

Test Macro	POSIX .1	POSIX .1a	POSIX .2	POSIX .4a	XPG 4.2	XPG 4.2 Ext
_POSIX_SOURCE	Yes					
_POSIX1_SOURCE 1	Yes					
_POSIX1_SOURCE 2	Yes	Yes				
_POSIX_C_SOURCE 1	Yes					
_POSIX_C_SOURCE 2	Yes		Yes			
_XOPEN_SOURCE	Yes		Yes		Yes	
XOPEN_SOURCE_EXTENDED 1	Yes		Yes		Yes	Yes
_OPEN_SYS	Yes	YES	Yes	Yes		
_OPEN_SYS_IPC_EXTENSIONs	Yes		Yes		Yes	
_OPEN_SYS_PTY_EXTENSIONs	Yes	YES	Yes		Yes	Yes

Table 13 (Page 2 of 2). Feature Test Macros and Standards						
Test Macro	**POSIX .1**	**POSIX .1a**	**POSIX .2**	**POSIX .4a**	**XPG 4.2**	**XPG 4.2 Ext**
_OPEN_THREADS	Yes	Yes		Yes		
_OPEN_SOURCE 1	Yes	Yes	Yes	Yes		
_XOPE_SOURCE 2 or _ALL_SOURCE	Yes	Yes	Yes	Yes	Yes	Yes
_OE_SOCKETS	Yes	Yes				
_OPEN_SOCKETS	Yes	Yes				

8.4.4 Compiler Options

We do not intend to describe all the available compiler options, but two options are of particular interest in this context:

LANGLVL(COMMONC)

This option is only available with C/MVS. It indicates language constructs defined by XPG, many of which are already supported in LANGLVL(EXTENDED). Exceptions to this are the following:

- Sign is preserved for standard integral promotions
- Trigraphs contained within literals are not processed
- *Sizeof* operator permitted on bit fields
- *Bit fields* other than int are permitted
- Macros within quotations are expanded
- There are no preconditions required for macro redefinition
- The empty comment in a function-like macro is equivalent to the ANSI/ISO token concatenation operator

Note: This option is available only with Version 3 Release 2 of the C/C++ Compiler for MVS/ESA.

Most often you have to use this compiler option due to bit fields defined within a byte (= char).

HWOPTS(STR)

The newer ES/9000 systems are provided with microcode to handle C strings more efficiently. Strings (char) are a collection of non-zero bytes which are terminated by a zero-byte (null-terminated strings). To move or copy a string from one place to another requires the ability to detect the end of the string.

By setting this option the compiler generates inline code based on the instructions provided with the newer ES/9000 systems. It is necessary to include the <strings.h> header file. Otherwise the compiler does *not* generate the inline code.

Note: This option is useful for programs that use strings a lot.

Compiler options may be passed either on the c89 command as shown below, or in makefiles.

```
c89 -W "0,langlvl(commonc)" -o Ernie Bert.c
```

8.5 Raw Socket Usage

Some TCP/IP applications use so-called *raw socket*. Raw sockets allow access to networking layer protocols (such as IP and ICMP). It is supported only in the AF_INET domain. It must be pointed out that usage of raw sockets is restricted to superusers. Consequently in OpenEdition you must have superuser authority to use such an application. As an example, we show the execution of *ping* in Figure 130 in both non-superuser (1) and superuser (2) mode.

```
pwd=/u/hdm/projects/oeping/ping: >ping mvs18a
ping: socket: EDC5139I Operation not permitted. 1
pwd=/u/hdm/projects/oeping/ping: >su
MEZ-1MESZ,086,268
/u/hdm/.setup script executed, uid=0(OMVSKERN) gid=1(OMVSGRP)
you have mail in /usr/mail/HDM.
pwd=/u/hdm/projects/oeping/ping: >ping -c 5 mvs18a 2
PING mvs18a.itso.ral.ibm.com (9.24.104.126): 56 data bytes
64 bytes from 9.24.104.126: icmp_seq=0 ttl=60 time=35.559 ms
64 bytes from 9.24.104.126: icmp_seq=1 ttl=60 time=4.745 ms
64 bytes from 9.24.104.126: icmp_seq=2 ttl=60 time=4.05 ms
64 bytes from 9.24.104.126: icmp_seq=3 ttl=60 time=4.049 ms
64 bytes from 9.24.104.126: icmp_seq=4 ttl=60 time=5.12 ms

--- mvs18a.itso.ral.ibm.com ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 4.049/10.704/35.559 ms
```

Figure 130. Execution of Ping in OpenEdition

Ping is useful to all users of OMVS, not only superusers. You can make ping available to all users of your OpenEdition system, if you create it as a *setuid* program, using the following procedure:

1. Switch to superuser mode.
2. Copy the ping program to a directory with world access, such as the /bin directory. Ping is now owned by UID(0).
3. Set the setuid bit via the following shell command:

   ```
   chmod 4755 ping
   ```

Ping is now available as a setuid program and all users on your OpenEdition system should be able to use it.

8.6 Time Distribution

In a distributed environment it is important to have a common understanding of what the current time is.

There are several implementations that address the time issue. The most refined ones are certainly the Distributed Time Service (DTS) of DCE (Distributed Computing Environment) and Network Time Protocol (NTP, RFC 1305). These are very precise, taking delays on network connections into account. A quite simple implementation is the *timed* implementation. The InetD server that is provided with OpenEdition implements a time daemon (see Figure 131 on page 202). The corresponding clients are implemented on various platforms. We tested a

version for OS/2 called *setclock*. When called with the *-s* option (see Figure 131 on page 202 **1**) the hardware clock of the PS/2 system is set. This allows for a simple adjustment of time, which may be sufficient in some cases.

```
[D:\]time
Current time is: 17:21:03.56
Enter the new time:

[D:\]setclock mvs18o
SETCLOCK Version 0.5   Apr 14 1996  - OS/2 TCP/IP Time Client
(c) IBM Corporation 1996.  All Rights Reserved.
Local  Time: Sat May 11 17:21:13 1996
Remote Time: Sat May 11 17:22:48 1996

[D:\]setclock mvs18o -s  1
SETCLOCK Version 0.5   Apr 14 1996  - OS/2 TCP/IP Time Client
(c) IBM Corporation 1996.  All Rights Reserved.
Local  Time: Sat May 11 17:21:18 1996
Remote Time: Sat May 11 17:22:53 1996
- Setting -
Local  Time: Sat May 11 17:22:53 1996
```

Figure 131. Timed Implementation of OpenEdition

Appendix A. Sample REXX to Create HOSTS.LOCAL from /etc/hosts

This sample REXX program converts an /etc/hosts local file into a HOSTS.LOCAL file and executes the TCP/IP for MVS MAKESITE utility to build the corresponding *datasetprefix*.HOSTS.ADDRINFO and *datasetprefix*.HOSTS.SITEINFO data sets.

```
/* REXX */
call syscalls 'ON'       /* Allow OMVS REXX services                */

hostfile = '/etc/hosts'  /* Input hosts file                        */
tcpiphlq = 'TCPIP.OMVS'  /* xxxxINFO data sets HLQ                  */
dotrace = 0              /* Set to 1 for application trace hooks    */

/* Let's start by reading the /etc/hosts file into a REXX stem      */

address syscall "readfile" hostfile "hostline."
if retval = -1 then do
  say hostfile 'not read, errno='errno' - errnojr='errnojr
  exit(8)
end
if hostline.0 = 0 then do
  say hostifle 'is empty - no lines read'
  exit(8)
end

/* Create or allocate existing &userid.HOSTS.LOCAL data set         */

If sysdsn(hosts.local) ¬= 'OK' then do
  address TSO "alloc fi(hostloc) da(hosts.local) lrecl(255),
  blksize(0) dsorg(ps) recfm(v b) new catalog"
  alcrc = rc
  If alcrc > 0 then do
    say 'Allocation of new 'userid()'.HOSTS.LOCAL failed - rc = 'alcrc
    exit(alcrc)
  end
end
else do
  address TSO "alloc fi(hostloc) da(hosts.local) shr"
end

/* Process /etc/hosts lines and create stem var for HOSTS.LOCAL     */

oix = 0
do i=1 to hostline.0
  If dotrace then say 'Trace input  ==> 'hostline.i
  if words(hostline.i) > 0 then do
    if left(word(hostline.i,1),1) = '#' then iterate
    if words(hostline.i) = 1 |,
    left(word(hostline.i,2),1) = '#' then do
      say 'The following line from 'hostfile' is not valid syntax:'
      say '==> 'hostline.i
      iterate
    end
    oix = oix+1
    outline.oix = 'HOST:'||word(hostline.i,1)||':'
    dropline = 0
    do x=2 to words(hostline.i)
      if left(word(hostline.i,x),1) = '#' then leave
```

© Copyright IBM Corp. 1996

```
      newdata = word(hostline.i,x)||','
      newlength = length(outline.oix) + length(newdata)
      if newlength+4 > 255 then do
        say 'The following output line is too long and will be dropped:'
        say '==> 'outline.oix
        dropline = 1
        leave
      end
      else do
        outline.oix = outline.oix||word(hostline.i,x)||','
      end
    end
    if dropline then
      oix = oix-1
    else do
      outline.oix=left(outline.oix,length(outline.oix)-1)
      outline.oix = outline.oix||'::::'
      If dotrace then say 'Trace output ==> 'outline.oix
    end /*Test for dropping line */
  end /* Input-line with data    */
end /* Do-loop for input lines   */
outline.0 = oix

/* Write lines to &userid.HOSTS.LOCAL and run MAKESITE               */

"EXECIO * DISKW HOSTLOC (STEM OUTLINE. FINIS"
address TSO "MAKESITE HLQ="tcpiphlq

exit(0)
```

Appendix B. Operations Automation Code Samples

The automation setup that was created during the creation of this book consists of NetView code, JCL procedures and OpenEdition shell scripts.

B.1 NetView Message Automation Table Entries

The automation is controlled via the NetView message automation table.

```
*************************************************************************
*        BEGINNING OF 'BPX' MSGID'S   - OE                              *
*************************************************************************
*
IF MSGID = 'BPX' . THEN BEGIN;
*
* BPXI004I OMVS Initialization has completed
* Start the default transport provider: T18OTCP
*
IF MSGID = 'BPXI004I' THEN BEGIN;
     IF DOMAINID = 'RAIAO'
     THEN EXEC( CMD('MVS S T18OTCP') ROUTE(ONE SYSAUTO))
          DISPLAY(Y) NETLOG(Y) SYSLOG(Y);
*
END;
*
END;
*
*************************************************************************
*        BEGINNING OF 'EZY' MSGID'S   - TCPIP                           *
*************************************************************************
*
IF MSGID = 'EZY' . THEN BEGIN;
*
* TCPIP Connection with OE established, re-cycle listener processes
* Must only match for the OE stack: T18OTCP
*
* EZY2140I OpenEdition-TCP/IP connection established for T18OTCP
* T: 1              2             3            4      5     6
*
IF MSGID = 'EZY2140I' & TOKEN(6) = 'T18OTCP' THEN BEGIN;
     IF DOMAINID = 'RAIAO'
     THEN EXEC( CMD('OERECYCL') ROUTE(ONE SYSAUTO))
          DISPLAY(Y) NETLOG(Y) SYSLOG(Y);
*
END;
*
END;
*
*************************************************************************
*        BEGINNING OF 'ISU' MSGID'S   - ANYNET MVS                      *
*************************************************************************
*
IF MSGID = 'ISU' . THEN BEGIN;
*
* AnyNet MVS is up, start AnyNet initialization job and
* re-cycle the OE Listener processes.
```

© Copyright IBM Corp. 1996

```
*
IF MSGID = 'ISU1501I' THEN BEGIN;
     IF DOMAINID = 'RAIAO'
     THEN EXEC( CMD('MVS S RAISOCKI') ROUTE(ONE SYSAUTO))
          EXEC( CMD('OERECYCL') ROUTE(ONE SYSAUTO))
        DISPLAY(Y) NETLOG(Y) SYSLOG(Y);
*
END;
*
END;
```

B.2 NetView REXX Program: OERECYCL

The NetView REXX program called OERECYCL controls the sequence in which the various listener processes are recycled. The OERECYCL REXX program calls other NetView REXX programs for the listener-specific recycle actions.

```
/* REXX */
/*********************************************************************/
/*                                                                   */
/* Name:       OERECYCL                                              */
/*                                                                   */
/* Function:   Netview REXX to re-cycle OE listener processes.       */
/*                                                                   */
/* Interface:  Started via message automation table.                 */
/*                                                                   */
/* Logic:      When a TCP/IP stack or an AnyNet stack has connected  */
/*             to OE, this REXX program is started.                  */
/*             It run various other Netview REXX programs to         */
/*             re-cycle the individual OE listener programs.         */
/*                                                                   */
/* Returncode: RC=0,  OK.                                            */
/*                                                                   */
/* Written:    May 1996, ITSO Raleigh                                */
/*                                                                   */
/* Modified:                                                         */
/*                                                                   */
/*********************************************************************/

Say '-- OERECYCL - Starting OERC to re-cycle SYSLOGD'
'OERC SYSLOGD'
myrc=lastcc
If myrc > 0 then Say '-- OERECYCL OERC Returncode='myrc

Say '-- OERECYCL - Starting OERC to re-cycle INETD'
'OERC INETD'
myrc=lastcc
If myrc > 0 then Say '-- OERECYCL OERC Returncode='myrc

Say '-- OERECYCL - Starting OEFTPD to re-cycle FTPD'
'OEFTPD'
myrc=lastcc
If myrc > 0 then Say '-- OERECYCL OEFTPD Returncode='myrc

Say '-- OERECYCL - Starting OEWEBS to re-cycle TCPW3SRV Web server'
'OEWEBS TCPW3SRV'
myrc=lastcc
If myrc > 0 then Say '-- OERECYCL OEWEBS Returncode='myrc
```

```
Say '-- OERECYCL - Starting OEWEBS to re-cycle CS2201SR Web server'
'OEWEBS CS2201SR'
myrc=lastcc
If myrc > 0 then Say '-- OERECYCL OEWEBS Returncode='myrc

exit(0)
```

B.3 NetView REXX Program: OERC

This NetView REXX program is used to recycle the SyslogD server and the InetD server. It is called from the OERECYCL REXX with the name of the server to recycle (either SYSLOGD or INETD).

This REXX program performs the following process:

1. If the server is running, an MVS start command for JCL procedure OERCSTOP is issued with the server name passed as argument SERVER. The OERC REXX program then loops with appropriate wait intervals to check if the server has come down or not. When the server is down, the processing continues to the next step.
2. Start the server via an MVS start command of JCL procedure OERCSTRT with the server name passed on the SERVER keyword.
3. If the server to recycle is SyslogD, the REXX program does not return to OERC before SyslogD has come up again. This is done to prevent us from starting other servers before SyslogD is running. If we start, for example, the FTPD server without SyslogD being up and running, the FTPD server messages all go to the MVS syslog, which can be a bit overwhelming if you have enabled any of the tracing options.

```
/* REXX */
/*********************************************************************/
/*                                                                   */
/* Name:       OERC                                                  */
/*                                                                   */
/* Function:   Netview REXX to re-cycle INETD or SYSLOGD.            */
/*                                                                   */
/* Interface:  Started via message automation table - server name    */
/*             is passed to this rexx (INETD or SYSLOGD).            */
/*                                                                   */
/* Logic:      When a TCP/IP stack or an AnyNet stack has connected  */
/*             to OE, this REXX program is started.                  */
/*             If old server is running, it is stopped via a started */
/*             task (OERCSTOP) that executes BPXBATCH to run a shell */
/*             script (stoprc) to stop the server.                   */
/*             When the server has stopped, it is started again by   */
/*             this REXX program via another started task (OERCSTRT) */
/*             that also runs BPXBATCH with a shell script           */
/*             (startrc) to start the server again.                  */
/*                                                                   */
/* Returncode: RC=0,  OK.                                            */
/*             RC=4,  Server started, but not up within 60 seconds.  */
/*             RC=8,  The REXX is called with an unknow server       */
/*                    name.                                          */
/*             RC=12, Old server did not come down within 30         */
/*                    seconds.  No new server is started.            */
/*                                                                   */
/* Written:    May 1996, ITSO Raleigh                                */
```

```
/*                                                                   */
/* Modified:                                                         */
/*                                                                   */
/*********************************************************************/

/*********************************************************************/
/*                                                                   */
/* The server is stopped via a shell script that finds the pid       */
/* and sends a signal to the server.  The shell script is executed   */
/* by submitting a BPXBATCH job (OERCSTOP).  We keep                 */
/* looping here until the server in question disappears from the     */
/* D OMVS output lines.                                              */
/*                                                                   */
/*********************************************************************/

parse upper arg servername

actrace = 1                             /* Application trace hooks   */
                                        /* 1: trace, 0: no-trace     */

stopissued = 0                          /* We only stop one time     */
SRVdead = 0                             /* Is server dead or not?    */
timeaccum = 0                           /* Max 30 seconds to stop    */

If actrace then
   say '-- OERC entered with servername='servername

Select
   When servername='INETD' then nop
   When servername='SYSLOGD' then nop
   Otherwise do
      say '-- Unknown server name: 'servername
      say '-- OERC stops.'
      exit(8)
   end
end

namelen = length(servername)

If actrace then
   Say '-- Entering loop to check for old server'

do until SRVdead
   If actrace then
      say '-- Doing D OMVS,U=OMVSKERN command'
   'pipe MVS D OMVS,U=OMVSKERN',
   '| corrwait 3',
   '| collect',
   '| stem result.'
   If actrace then
      say '-- Number of display output lines = 'result.0
   If result.0 > 0 then do
      SRVdead = 1
      do i=1 to result.0
         parse var result.i userid jobname . . ppid .
         If substr(jobname,1,namelen)=servername then do
            if userid='OMVSKERN' & ppid=1 then do
               SRVdead = 0
               if actrace then
```

```
                  say '-- Found server'
               if stopissued then do
                  delay 5
                  timeaccum=timeaccum+5
               end
               else do
                  Say '-- Netview OERC: Stopping old server - S OERCSTOP'
                  'MVS S OERCSTOP,SERVER='servername
                  stopissued = 1
                  delay 5
                  timeaccum=timeaccum+5
               end
            end /* Server was there */
         end /* Name match          */
      end /* result line loop       */
   end /* OK D OMVS command         */
   else do
      say '-- No response from D OMVS command'
      delay 2
      timeaccum = timeaccum + 2
   end
   If timeaccum > 45 then do
      say '-- 45 seconds since we stopped 'servername', still not down.'
      say '-- We stop and exit without restarting server.'
      say '-- You may have to restart 'servername' manually.'
      exit(12)
   end
end /* loop until no server  */

/*********************************************************************/
/*                                                                   */
/* Now we are ready to start the server again                        */
/*                                                                   */
/*********************************************************************/

Say '-- Netview OERC: Starting 'servername

'MVS S OERCSTRT,SERVER='servername

/*********************************************************************/
/*                                                                   */
/* If we started SYSLOGD, do not exit until SYSLOGD is up and        */
/* running again.                                                    */
/*                                                                   */
/*********************************************************************/


If servername = 'SYSLOGD' then do
   timeaccum = 0
   SRValive = 0
   If actrace then
      say '-- Starting to test for' servername 'up again.'
   Do until SRValive
      If actrace then
         say '-- Doing D OMVS,U=OMVSKERN command'
      'pipe MVS D OMVS,U=OMVSKERN',
      '| corrwait 3',
      '| collect',
      '| stem result.'
```

```
      If actrace then
         say '-- Number of display output lines = 'result.0
      If result.0 > 0 then do
         do i=1 to result.0
            parse var result.i userid jobname . . ppid .
            If substr(jobname,1,namelen)=servername then do
               if userid='OMVSKERN' & ppid=1 then do
                  SRValive = 1
                  If actrace then
                     say '-- 'servername' is up again.'
                  leave
               end /* Syslogd match */
            end /* Jobname match    */
         end /* Command line loop   */
      end /* D OMVS output lines    */
      else do
         say '-- No response from D OMVS command'
         delay 5
         timeaccum = timeaccum + 5
      end
      If SRValive then leave
      If actrace then
         Say '-- 'servername' not up yet..'
      If timeaccum > 60 then do
         say '-- 60 seconds since we started 'servername', still not up.'
         say '-- We stop and exit now.'
         exit(4)
      end
      Delay 15
      timeaccum = timeaccum + 15
   end /* Do until syslogd is up    */
end /* SYSLOGD server               */

If actrace then
   say '-- Exiting after having started 'servername

exit(0)
```

B.4 JCL Procedure: OERCSTOP

This JCL procedure starts a BPXBATCH program that starts an OpenEdition shell REXX program called stoprc. The stoprc REXX program accepts one argument: the name of the server to stop.

```
//OERCSTOP PROC SERVER=
//*
//* Run BPXBATCH to stop SYSLOGD or INETD
//*
//SHELL EXEC PGM=BPXBATCH,REGION=4OM,
// PARM='SH /u/alfredc/stoprc &SERVER.'
//STDOUT  DD PATH='/u/alfredc/oercstop.stdout',
//           PATHMODE=SIRWXU,PATHOPTS=(OWRONLY,OCREAT)
//STDERR  DD PATH='/u/alfredc/oercstop.stderr',
//           PATHMODE=SIRWXU,PATHOPTS=(OWRONLY,OCREAT)
//*
//* Copy stdout and stderr to sysprint
//*
//COPYPRT  EXEC PGM=IKJEFT01,DYNAMNBR=2O
//SYSTSPRT DD SYSOUT=*
```

```
//SYSOUT   DD SYSOUT=*
//STDOUT   DD PATH='/u/alfredc/oercstop.stdout',
//            PATHOPTS=(ORDONLY)
//STDERR   DD PATH='/u/alfredc/oercstop.stderr',
//            PATHOPTS=(ORDONLY)
//SYSPRINT DD SYSOUT=*
//HFSOUT   DD SYSOUT=*,DCB=(RECFM=V,LRECL=256)
//SYSTSIN  DD DSN=TCPIP.ITSC.CNTL(OCOPY),DISP=SHR
//         PEND
```

B.5 OpenEdition Shell REXX Program: stoprc

This REXX shell program uses the server name as an argument. It uses a ps -ef command to determine the process ID of the server, and then sends a kill signal to the server.

```
/* REXX */
/*********************************************************************/
/*                                                                   */
/* Name:       stoprc                                                */
/*                                                                   */
/* Function:   OE shell REXX to stop INETD or SYSLOGD                */
/*                                                                   */
/* Interface:  Started via BPXBATCH started task: OERCSTOP with      */
/*             servername as argument (INETD or SYSLOGD).            */
/*                                                                   */
/* Logic:      This REXX is kicked off from the OERC NetView REXX    */
/*             that is started via message automation whenever we    */
/*             need to restart INETD or SYSLOGD.                     */
/*             This REXX sends a kill signal to the currently        */
/*             executing server and exits.  The OERC NetView REXX    */
/*             tests when the server is actually down, and starts    */
/*             another procedure: OERCSTRT to start the server again */
/*             through another BPXBATCH execution.                   */
/*                                                                   */
/* Returncode: RC=0,  OK.                                            */
/*                                                                   */
/* Written:    May 1996, ITSO Raleigh                                */
/*                                                                   */
/* Modified:                                                         */
/*                                                                   */
/*********************************************************************/
/* ************************************************* */
/*                                                   */
/* Experience shows us that pid files do not always  */
/* hold the process IDs.  Sometimes they are just    */
/* empty files.  To find the process id of the current */
/* processes, we issue a ps -ef command and analyze  */
/* the output lines.                                 */
/*                                                   */
/* Output from ps -ef sometimes has a date and not a */
/* time in the time column, which makes parsing a bit */
/* interesting.  Before we parse the command name, we */
/* strip off the first 58 characters of the line.    */
/*                                                   */
/*          **PID**                  ****CMD******** */
/* OMVSKERN 2162698 1 - 17:03:07 ? 0:00 /usr/sbin/InetD */
/*                                   col 59          */
/*                                                   */
```

```
/* ************************************************** */
parse arg server
uppertbl = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
lowertbl = 'abcdefghijklmnopqrstuvwxyz'
serverprog = translate(server,lowertbl,uppertbl) /* To lowercase */
Say '-- Invoked to stop 'serverprog
address syscall 'pipe p.'
'ps -ef>/dev/fd'||p.2
address syscall 'close' p.2
address mvs 'execio * diskr' p.1 '(stem s.'
do i=1 to s.0
  parse var s.i user pid ppid .
  cmdstring = substr(s.i,59,21)
  parse var cmdstring cmd .
  if cmd = '/usr/sbin/'||serverprog & ppid=1 then do
    say '-- Killing old 'serverprog' (Old PID='pid')'
    address syscall 'kill 'pid SIGTERM
  end
end
exit(0)
```

B.6 JCL Procedure: OERCSTRT

This JCL procedure starts a BPXBATCH program that starts an OpenEdition shell REXX program called startrc. The startrc REXX program accepts one argument: the name of the server to start.

```
//OERCSTRT PROC SERVER=
//*
//* Run BPXBATCH to start SYSLOGD or INETD
//*
//SHELL EXEC PGM=BPXBATCH,REGION=4OM,
// PARM='SH /u/alfredc/startrc &SERVER.'
//STDOUT  DD PATH='/u/alfredc/oercstrt.stdout',
//           PATHMODE=SIRWXU,PATHOPTS=(OWRONLY,OCREAT)
//STDERR  DD PATH='/u/alfredc/oercstrt.stderr',
//           PATHMODE=SIRWXU,PATHOPTS=(OWRONLY,OCREAT)
//*
//* Copy stdout and stderr to sysprint
//*
//COPYPRT  EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//STDOUT   DD PATH='/u/alfredc/oercstrt.stdout',
//            PATHOPTS=(ORDONLY)
//STDERR   DD PATH='/u/alfredc/oercstrt.stderr',
//            PATHOPTS=(ORDONLY)
//SYSPRINT DD SYSOUT=*
//HFSOUT   DD SYSOUT=*,DCB=(RECFM=V,LRECL=256)
//SYSTSIN  DD DSN=TCPIP.ITSC.CNTL(OCOPY),DISP=SHR
//         PEND
```

B.7 OpenEdition Shell REXX Program: startrc

This REXX shell program uses the server name as an argument. If the server is InetD, it starts a shell script called restart_inetd.sh. If the server is SyslogD, it starts another shell script called restart_syslogd.sh. We need to start the servers from a shell script, because we are not able to set new environment variables in a REXX shell program, and we need to set the BPXJOBNAME environment variable in order to assign the proper job names to the InetD and SyslogD processes.

```
/* REXX */
/*********************************************************************/
/*                                                                   */
/* Name:       startrc                                               */
/*                                                                   */
/* Function:   OE shell REXX to start INETD or SYSLOGD               */
/*                                                                   */
/* Interface:  Started via BPXBATCH started task: OERCSTRT with      */
/*             servername as argument (INETD or SYSLOGD).            */
/*                                                                   */
/* Logic:      This REXX is kicked off from the OERC Netview REXX    */
/*             that is started via message automation whenever we    */
/*             need to restart INETD or SYSLOGD.                     */
/*             This REXX starts either INETD or SYSLOGD based on     */
/*             invocation argument.                                  */
/*                                                                   */
/* Returncode: RC=0,  OK.                                            */
/*                                                                   */
/* Written:    May 1996, ITSO Raleigh                                */
/*                                                                   */
/* Modified:                                                         */
/*                                                                   */
/*********************************************************************/
parse arg server
uppertbl = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
lowertbl = 'abcdefghijklmnopqrstuvwxyz'
serverprog = translate(server,lowertbl,uppertbl) /* To lowercase */
Say '-- Invoked to start 'serverprog
/* ************************************************** */
/*                                                    */
/* Servers are started via small shell scripts that   */
/* set the _BPX_JOBNAME environment variable before   */
/* starting server, so MVS jobname is correct (INETDn */
/* or SYSLOGDn)                                       */
/*                                                    */
/* ************************************************** */
Select
  When serverprog = 'syslogd' then do
     Say '-- Starting SyslogD'
     address SH '/u/alfredc/restart_syslogd.sh'
     Say '-- Start of SyslogD done..'
  end
  When serverprog = 'inetd' then do
     Say '-- Starting InetD'
     address SH '/u/alfredc/restart_inetd.sh'
     Say '-- Start of InetD done..'
  end
  Otherwise do
     Say '-- Unknown server name: 'serverprog' - not started.'
```

```
    exit(8)
  end
end
exit(0)
```

B.8 Shell Script: restart_inetd.sh

This shell script is used to start the InetD process.

```
# Start the INETD daemon
#
# This shell script is invoked from the startrc REXX program
#
# export _CEE_RUNOPTS='TRACE(ON,1M,DUMP,LE=1) ALL31(ON)'
export _BPX_JOBNAME='INETD'
export _CEE_RUNOPTS='ALL31(ON)'
/usr/sbin/inetd /etc/inetd.conf &
echo -- /u/alfredc/restart_inetd.sh script executed, date
```

B.9 Shell Script: restart_syslogd.sh

This shell script is used to start the SyslogD process.

```
# Start the SYSLOGD daemon
#
# This shell script is invoked from the startrc REXX program
#
# export _CEE_RUNOPTS='TRACE(ON,1M,DUMP,LE=1) ALL31(ON)'
export _BPX_JOBNAME='SYSLOGD'
export _CEE_RUNOPTS='ALL31(ON)'
/usr/sbin/syslogd -f /etc/syslog.conf &
echo -- /u/alfredc/restart_syslogd.sh script executed, date
```

B.10 NetView REXX Program: OEFTPD

This NetView REXX program is used to recycle the FTPD listener process. If one is already running, it is stopped. The REXX program waits until the currently executing server has terminated and then it starts it again. Both the stop and start are done via normal MVS operator commands.

```
/* REXX */
/*********************************************************************/
/*                                                                   */
/* Name:       OEFTPD                                                */
/*                                                                   */
/* Function:   NetView REXX to re-cycle OE FTP server                */
/*                                                                   */
/* Interface:  Started via message automation table - no parms       */
/*                                                                   */
/* Logic:      When a TCP/IP stack or an AnyNet stack has connected  */
/*             to OE, this REXX program is started.                  */
/*             If an old FTPD1 process is running, it is stopped.    */
/*             FTPD is then started.                                 */
/*                                                                   */
/* Returncode: RC=0,  OK.                                            */
/*             RC=8,  Old FTPD1 did not come down within 30          */
/*                    seconds.  No new FTPD is started.              */
/*                                                                   */
```

```
/* Written:    May 1996, ITSO Raleigh                                */
/*                                                                   */
/* Modified:                                                         */
/*                                                                   */
/*********************************************************************/

/*********************************************************************/
/*                                                                   */
/* If an FTPD1 job is already running, we stop it and wait until     */
/* it has closed down, before we continue to start it.               */
/* We recognize the listener via user ID OMVSKERN and jobname FTPD1. */
/*                                                                   */
/* The check for FTPD1 includes a check of parent PID = 1.  If we    */
/* have 10 connected clients, one of them might have gotten the      */
/* jobname FTPD1 and might even be running under user ID OMVSKERN -  */
/* if he just connected, but did not enter user ID and password yet. */
/* Only the listener FTPD1 will run with parent PID of 1.            */
/*                                                                   */
/* When we stop the old FTPD1, we use the asid from D OMVS command   */
/* to ensure that the stop command goes to the right FTPD1 jobname.  */
/*                                                                   */
/*********************************************************************/

actrace = 0                             /* Application trace hooks   */
                                        /* 1: trace, 0: no-trace     */

stopissued = 0                          /* We only stop one time     */
FTPD1dead = 0                           /* Is FTPD1 dead or not?     */
timeaccum = 0                           /* Max 30 seconds to stop    */

If actrace then
   Say '-- Entering loop to check for old FTPD1'

do until FTPD1dead
   If actrace then
      say '-- Doing D OMVS,U=OMVSKERN command'
   'pipe MVS D OMVS,U=OMVSKERN',
   '| corrwait 3',
   '| collect',
   '| stem result.'
   If actrace then
      say '-- Number of display output lines = 'result.0
   If result.0 > 0 then do
      FTPD1dead = 1
      do i=1 to result.0
         If actrace then
            say '-- 'result.i
         parse var result.i userid jobname asid . ppid .
         if userid='OMVSKERN' & jobname='FTPD1' & ppid=1 then do
            FTPD1dead = 0
            if actrace then
               say '-- Found FTPD1'
            if stopissued then do
               delay 5
               timeaccum = timeaccum + 5
            end
            else do
               Say '-- Netview OEFTPD: Stopping old FTPD1'
               'MVS P FTPD1,A='asid
```

```
                stopissued = 1
                delay 5
                timeaccum = timeaccum + 5
             end
          end /* FTPD1 was there */
       end /* result line loop   */
    end /* OK D OMVS command     */
    else do
       say '-- No response from D OMVS command'
       delay 2
       timeaccum = timeaccum + 2
    end
    If timeaccum > 30 then do
       say '-- 30 seconds since we issued P FTPD1, still not down -'
       say '-- We stop and exit without restarting FTPD.'
       exit(8)
    end
end /* loop until no FTPD1   */

/*********************************************************************/
/*                                                                   */
/* Now we are ready to start FTPD.                                   */
/*                                                                   */
/*********************************************************************/

Say '-- NetView OEFTPD: Starting FTPD'

'MVS S FTPD'

If actrace then
   say '-- Exiting after having started FTPD'

exit(0)
```

B.11 NetView REXX Program: OEWEBS

This NetView REXX program controls the recycle of a Web server. The name of the server is passed as an argument to the REXX program. It can have one of two values in the ITSO sample setup, TCPW3SRV or CS2201SR, which were the two Web server instances that were used in the ITSO environment.

If the server name is TCPW3SRV, the REXX program does the following:

1. Issues an MVS start command for JCL procedure OETCPW3.
2. Waits for the server to disappear.
3. Issues an MVS start command for JCL procedure TCPW3SRV to start the server again.

If the server name is CS2201SR, the REXX program does the following:

1. Issues an MVS start command for JCL procedure OECS2201.
2. Waits for the server to disappear.
3. Issues an MVS start command for JCL procedure CS2201SR to start the server again.

```
/* REXX */
/*********************************************************************/
/*                                                                   */
/* Name:       OEWEBS                                                */
/*                                                                   */
/* Function:   NetView REXX to re-cycle an MVS WEB server.           */
/*                                                                   */
/* Interface:  Started via message automation table - server name    */
/*             is passed to this rexx (TCPW3SRV or CS2201SR).        */
/*                                                                   */
/* Logic:      When a TCP/IP stack or an AnyNet stack has connected  */
/*             to OE, this REXX program is started.                  */
/*             If old Web server is running, it is stopped via       */
/*             a started task that executes BPXBATCH to run a shell  */
/*             command to stop the server.                           */
/*             When the server has stopped, it is started again by   */
/*             this REXX program via an MVS start command.           */
/*                                                                   */
/* Returncode: RC=0,  OK.                                            */
/*             RC=8,  The REXX is called with an unknow server       */
/*                    name.                                          */
/*             RC=12, Old server did not come down within 2 minutes  */
/*                    seconds.  No new server is started.            */
/*                                                                   */
/* Written:    May 1996, ITSO Raleigh                                */
/*                                                                   */
/* Modified:                                                         */
/*                                                                   */
/*********************************************************************/

/*********************************************************************/
/*                                                                   */
/* The Web server is stopped via shell scripts that execute the      */
/* wwwcmd program that comes with the Web server.  These shell       */
/* scripts are executed by submitting a BPXBATCH job.  We keep       */
/* looping here until the server in question disappears from the     */
/* D OMVS output lines.                                              */
/*                                                                   */
/* We will allow old Web server up to two minutes to stop, before    */
/* we give up waiting for it.                                        */
/*                                                                   */
/*********************************************************************/

parse upper arg servername

actrace = 0                            /* Application trace hooks   */
                                       /* 1: trace, 0: no-trace     */

stopissued = 0                         /* We only stop one time     */
SRVdead = 0                            /* Is server dead or not?    */
timeaccum = 0                          /* Max 30 seconds to stop    */

If actrace then
   say '-- OEWEBS entered with servername='servername

Select
   When servername='TCPW3SRV' then do
      stuser='TCPW3SRV'                /* TCPW3SRV STC userID       */
      proc='OETCPW3'                   /* Procedure to stop TCPW3SRV*/
```

```
   end
   When servername='CS2201SR' then do
      stuser='CS2201SR'                  /* CS2201SR STC userID       */
      proc='OECS2201'                    /* Procedure to stop CS2201SR*/
   end
   Otherwise do
      say '-- Unknown server name: 'servername
      say '-- OEWEBS stops.'
      exit(8)
   end
end

If actrace then
   Say '-- Entering loop to check for old server'

do until SRVdead
   If actrace then
      say '-- Doing D OMVS,U='stuser' command'
   'pipe MVS D OMVS,U='stuser,
   '| corrwait 3',
   '| collect',
   '| stem result.'
   If actrace then
      say '-- Number of display output lines = 'result.0
   If result.0 > 0 then do
      SRVdead = 1
      do i=1 to result.0
         If actrace then
            say '-- 'result.i
         parse var result.i userid jobname . . ppid .
         if jobname=servername & userid=stuser & ppid=1 then do
            SRVdead = 0
            if actrace then
               say '-- Found server'
            if stopissued then do
               delay 15
               timeaccum=timeaccum+15
            end
            else do
               Say '-- Netview OEWEBS: Stopping old server - S 'proc
               'MVS S 'proc
               stopissued = 1
               delay 15
               timeaccum=timeaccum+15
            end
         end /* Server was there */
      end /* result line loop    */
   end /* OK D OMVS command      */
   else do
      say '-- No response from D OMVS command'
      delay 5
      timeaccum = timeaccum + 5
   end
   If timeaccum > 120 then do
      say '-- 2 minutes since we stopped 'servername', still not down..'
      say '-- We stop and exit without restarting server.'
      say '-- You may have to restart 'servername' manually.'
      exit(12)
```

```
   end
end /* loop until no server  */

/*********************************************************************/
/*                                                                   */
/* Now we are ready to start the Web server                          */
/*                                                                   */
/*********************************************************************/

Say '-- Netview OEWEBS: Starting 'servername

'MVS S 'servername

If actrace then
   say '-- Exiting after having started 'servername

exit(0)
```

B.12 JCL Procedure: OETCPW3

This JCL procedure runs a BPXBATCH program that executes an OpenEdition shell REXX program called stopweb_tcpw3srv.

```
//OETCPW3 PROC
//*
//* Stop TCPW3SRV Web server
//*
//* Run BPXBATCH to stop Web Server
//*
//SHELL EXEC PGM=BPXBATCH,REGION=4OM,
// PARM='SH /u/alfredc/stopweb_tcpw3srv'
//STDOUT  DD PATH='/u/alfredc/stopweb_tcpw3srv.stdout',
//           PATHMODE=SIRWXU,PATHOPTS=(OWRONLY,OCREAT)
//STDERR  DD PATH='/u/alfredc/stopweb_tcpw3srv.stderr',
//           PATHMODE=SIRWXU,PATHOPTS=(OWRONLY,OCREAT)
//*
//* Copy stdout and stderr to sysprint
//*
//COPYPRT  EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//STDOUT   DD PATH='/u/alfredc/stopweb_tcpw3srv.stdout',
//            PATHOPTS=(ORDONLY)
//STDERR   DD PATH='/u/alfredc/stopweb_tcpw3srv.stderr',
//            PATHOPTS=(ORDONLY)
//SYSPRINT DD SYSOUT=*
//HFSOUT   DD SYSOUT=*,DCB=(RECFM=V,LRECL=256)
//SYSTSIN  DD DSN=TCPIP.ITSC.CNTL(OCOPY),DISP=SHR
//         PEND
```

B.13 OpenEdition Shell REXX Program: stopweb_tcpw3srv

This shell REXX program executes the wwwcmd program that comes with the Internet Connection Server for MVS. The wwwcmd program is executed with parameters pointing to the TCPW3SRV root directory where the httpd.pid file for the server is located.

```
/* REXX */
/* ************************************************** */
/* REXX to stop running Web servers.                  */
/*                                                    */
/* This REXX program stops the following Web server:  */
/*  TCPW3SRV (pid in /u/tcpipweb/docroot/hhtpd.pid)   */
/*                                                    */
/* ************************************************** */
address SH '/u/tcpipweb/docroot/cgi-bin/wwwcmd -f /u/tcpipweb/docroot/httpd-pid -stop'
exit(0)
```

B.14 JCL Procedure: OECS2201

This JCL procedure runs a BPXBATCH program that executes an OpenEdition shell REXX program called stopweb_cs2201sr.

```
//OECS2201 PROC
//*
//* Stop CS2201SR Web server
//*
//* Run BPXBATCH to stop Web Server
//*
//SHELL EXEC PGM=BPXBATCH,REGION=40M,
// PARM='SH /u/alfredc/stopweb_cs2201sr'
//STDOUT  DD PATH='/u/alfredc/stopweb_cs2201sr.stdout',
//           PATHMODE=SIRWXU,PATHOPTS=(OWRONLY,OCREAT)
//STDERR  DD PATH='/u/alfredc/stopweb_cs2201sr.stderr',
//           PATHMODE=SIRWXU,PATHOPTS=(OWRONLY,OCREAT)
//*
//* Copy stdout and stderr to sysprint
//*
//COPYPRT  EXEC PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//STDOUT   DD PATH='/u/alfredc/stopweb_cs2201sr.stdout',
//            PATHOPTS=(ORDONLY)
//STDERR   DD PATH='/u/alfredc/stopweb_cs2201sr.stderr',
//            PATHOPTS=(ORDONLY)
//SYSPRINT DD SYSOUT=*
//HFSOUT   DD SYSOUT=*,DCB=(RECFM=V,LRECL=256)
//SYSTSIN  DD DSN=TCPIP.ITSC.CNTL(OCOPY),DISP=SHR
//         PEND
```

B.15 OpenEdition Shell REXX Program: stopweb_cs2201sr

This shell REXX program executes the wwwcmd program that comes with the Internet Connection Server for MVS. The wwwcmd program is executed with parameters pointing to the CS2201SR root directory where the httpd.pid file for the server is located.

```
/* REXX */
/* ************************************************** */
/* REXX to stop running Web servers.                  */
/*                                                    */
/* This REXX program stops the following Web server:  */
/*  CS2201SR (pid in /u/cs2201/docroot/httpd.pid)     */
/*                                                    */
/* ************************************************** */
address SH '/u/tcpipweb/docroot/cgi-bin/wwwcmd -f /u/cs2201/docroot/httpd-pid -stop'
exit(0)
```

Appendix C. Sample Curses Application

The following is the C program we used to test *curses*. The output of it, though boring, is shown in Figure 132 on page 222.

Note: We left out the clearscreen call to get the screen captured.

```
#include <curses.h>
#include <term.h>
#include <stdio.h>
#include <stdlib.h>

main(int argc, char *argv[], char *envp[])
{
   char * term ;                  /* the terminal name */
   int status ;
   int i,j ;
   char * s, zz[3] ;

   /* get the teminal type */
   if ((term = getenv("TERM")) == NULL) {
      fatal("Could not locate TERM environment variable\n");
   } /* endif */
   printf("\nTERM=%s",term);

   setupterm(term, 1, &status);
   if (status!= 1) {
      fatal("Error in setupterm\n");
   } /* endif */

   /* clear the screen */
   if (clear_screen == NULL) {
      fatal("clear string is not set\n");
   } /* endif */
   putp(clear_screen);           /* do it */
   if (cursor_address == NULL ) {
      fatal("cursor motion string is not set\n") ;
   } /* endif */
   for (i=0;i < lines ;i++ ) {
      for (j=0;j < columns ;j++ ) {
         if (i==lines-1 && j==columns-1) continue ;
         s = tparm(cursor_address, i, j) ;
         putp(s);
         if (j==0) { /* we are on col 0 , show current line # */
         zz[0] = '\0' ;
         sprintf(zz,"%2d",i+1);
         putchar(zz[0]);
         j++; /* needed room for 2 chars */
         s = tparm(cursor_address, i, j) ;
         putchar(zz[1]);
         } else
         putchar('*');
      } /* endfor columns */
   } /* endfor lines */
   putp(clear_screen) ;
   exit(EXIT_SUCCESS) ;
}

fatal( char * message )
{
   fprintf (stderr, "%s\n", message) ;
   exit(EXIT_FAILURE);
}
```

© Copyright IBM Corp. 1996

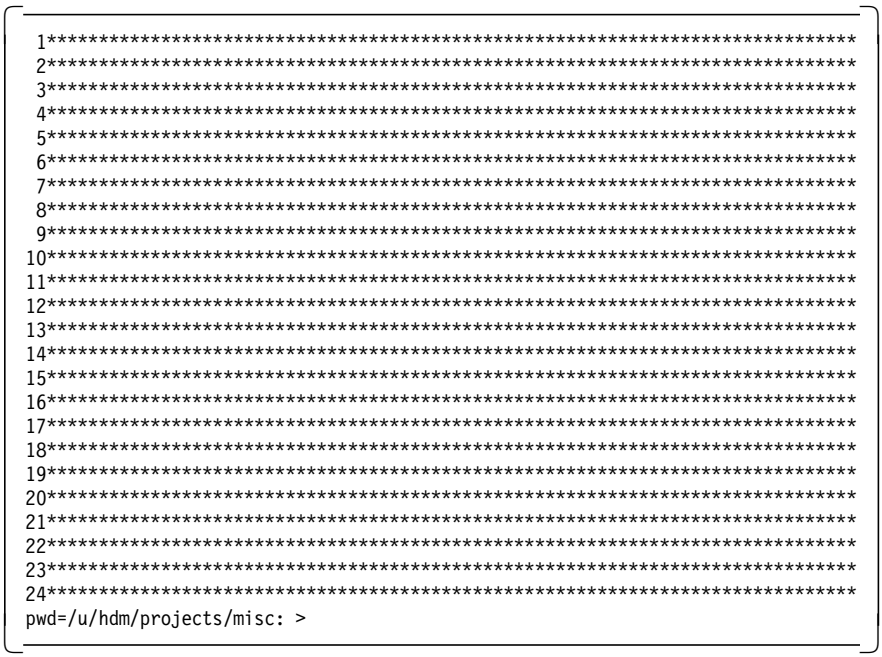


Figure 132. Output From Example Program Using Curses

Appendix D. Sample ONC/RPC Application

This appendix contains the C source code of the sample ONC/RPC application that was used to test the ONC/RPC component of the TCP/IP for MVS OpenEdition Applications Feature.

D.1 Readdir.c

```
/* /u/hdm/rpc/rls/manual/minimum/read_dir.c
   note - RPC compliant procedure calls take one input and
   return one output. Everything is passed by pointer.
   Return values should point to static data, as it might
   have to survive some while. */
#ifdef MVS
/* #define _OE_SOCKETS
#define _XOPEN_SOURCE_EXTENDED 1 */
#define _ALL_SOURCE
#endif
#include <stdio.h>
#include <sys/types.h>
/* use
<xpg2include/sys/dirent.h>     (SunOS4.1)  or
<sys/dirent.h> for X/Open Portability Guide, 4.2 conformance */
#ifdef  MVS
#include <sys/dirent.h>
#else
#include <sys/dir.h>
#endif
#include "rls.h"

int read_dir(dir)
char    *dir;   /* char dir[DIR_SIZE] */
{
        DIR * dirp;
#ifdef  MVS
        struct dirent *d;
#else
        struct direct *d;
#endif
        printf("beginning ");

        /* open directory */
        dirp = opendir(dir);
        if (dirp == NULL)
                return((int)NULL);

        /* stuff filenames into dir buffer */
        dir[0] = '\0' /* NULL*/ ;
        while (d = readdir(dirp))
                sprintf(dir, "%s%s\n", dir, d->d_name);

        /* return the result */
        printf("returning ");
        closedir(dirp);
        return((int)dir);  /* this is the only new line from Example 4-3 */
}
```

D.2 Rls_svc.c

```
/* /u/hdm/rpc/rls/manual/minimum/rls_svc.c
*/
#ifdef MVS
/*
#define _OE_SOCKETS
#define _XOPEN_SOURCE_EXTENDED 1 */
#define _ALL_SOURCE
#endif
```

© Copyright IBM Corp. 1996

```
#include <rpc/rpc.h>
#include "rls.h"

main()
{
        extern bool_t xdr_dir();
        extern char * read_dir();

        registerrpc(DIRPROG, DIRVERS, READDIR,
                        read_dir, xdr_dir, xdr_dir);

        svc_run();
}
```

D.3 Rls.c

```
/*
 * /u/hdm/rpc/rls/manual/minimum/rls.c
 * rls.c: remote directory listing client
 */
#ifdef MVS
/*
#define _OE_SOCKETS
#define _XOPEN_SOURCE_EXTENDED 1 */
#define _ALL_SOURCE
#endif
#include <stdio.h>
#include <strings.h>
#include <rpc/rpc.h>
#include "rls.h"

main (argc, argv)
int argc; char *argv[];
{
char    dir[DIR_SIZE];

        /* call the remote procedure if registered */
        strcpy(dir, argv[2]);
        read_dir(argv[1], dir); /* read_dir(host, directory) */

        /* spew-out the results and bail out of here! */
        printf("%s\n", dir);

        exit(0);
}

read_dir(host, dir)
char   *dir, *host;
{
        extern bool_t xdr_dir();
        enum clnt_stat clnt_stat;

        clnt_stat = callrpc ( host, DIRPROG, DIRVERS, READDIR,
                        xdr_dir, dir, xdr_dir, dir);
        if (clnt_stat != 0) clnt_perrno (clnt_stat);
}
```

D.4 Lls.c

```
/*
 * /u/hdm/rpc/rls/manual/minimum/lls.c
 * ls.c: local directory listing main - before RPC
 */
#include <stdio.h>
#include <strings.h>
#include "rls.h"

main (argc, argv)
int argc; char *argv[];
{
char    dir[DIR_SIZE];
```

```
        /* call the local procedure */
        strcpy(dir, argv[1]) /* char dir[DIR_SIZE]
                        is coming and going...*/
        read_dir(dir);

        /* spew-out the results and bail out of here! */
        printf("%s\n", dir);

        exit(0);
}
```

D.5 TcpRls.c

```
/* /u/hdm/rpc/rls/manual/minimum/tcpRls.c
*/
#ifdef MVS
/*
#define _OE_SOCKETS
#define _XOPEN_SOURCE_EXTENDED 1 */
#define _ALL_SOURCE
#endif
/*
 * rls.c: remote directory listing client
 */
#include <stdio.h>
#include <strings.h>
#include <rpc/rpc.h>
#include "rls.h"

main(argc, argv)
  int             argc;
  char           *argv[];
{
  char            d[DIR_SIZE];

  /* call the remote procedure if registered */
  strcpy(d, argv[2]);
  read_dir(argv[1], d); /* read_dir(host, directory) */

  /* spew-out the results and bail out of here! */
  printf("%s\n", d);

  exit(0);
}

read_dir(host, d)
  char           *d, *host;
{
  extern bool_t   xdr_dir();
  enum clnt_stat  clnt_stat;
  static struct timeval TIMEOUT = {25, 0};
  CLIENT         *clnt;

  clnt = clnt_create(host, DIRPROG, DIRVERS, "tcp");
  clnt_stat = clnt_call(clnt, READDIR,
                xdr_dir, d, xdr_dir, d, TIMEOUT);
  if (clnt_stat != 0)
    clnt_perrno(clnt_stat);
  clnt_destroy(clnt);
}
```

D.6 Rls.h

```
#define DIR_SIZE 8192
#define DIRPROG ((u_long) 0x20000001) /* server program (suite) number */
#define DIRVERS ((u_long) 1)    /* program version number */
#define READDIR ((u_long) 1)    /* procedure number for look-up */
```

Appendix E. Configuration File Samples

This appendix contains the majority of configuration files or data sets that were used in the sample ITSO-Raleigh setup.

The following groups of samples are included:

- E.1, TCP/IP for MVS Configuration Files for OE Stack
- E.2, OpenEdition Configuration and Files
- E.3, AnyNet MVS Configuration Data Sets

E.1 TCP/IP for MVS Configuration Files for OE Stack

The TCP/IP for MVS OE stack in our setup uses a DATASETPREFIX of TCPIP.T18OE. The following data sets exist with this high-level qualifier:

```
TCPIP.T18OE.ETC.PROTO
TCPIP.T18OE.ETC.RPC
TCPIP.T18OE.ETC.SERVICES
TCPIP.T18OE.HOSTS.ADDRINFO
TCPIP.T18OE.HOSTS.SITEINFO
TCPIP.T18OE.STANDARD.TCPXLBIN
TCPIP.T18OE.TCPPARMS
```

The ETC.PROTO, ETC.RPC and ETC.SERVICES data sets all have the standard contents without any modifications.

The HOSTS.ADDRINFO and HOSTS.SITEINFO data sets are maintained based on the /etc/hosts file content. These data sets are updated by the REXX program we described in Appendix A, “Sample REXX to Create HOSTS.LOCAL from /etc/hosts” on page 203.

The TCPPARMS data set is an MVS PDS and it contains most of the configurations that are used by this TCP/IP for MVS stack. The members of this library are:

```
GATEWAYS  - RouteD GATEWAYS specifications
NSCACHE   - Caching-only name server cache info
NSMAIN    - Caching-only name server configuration
PROFILE   - System address space PROFILE.TCPIP
TCPDATA   - TCPIP.DATA for this stack
```

The above members are all documented in this appendix along with various JCL procedures.

E.1.1 T18OTCP - OE Stack TCP/IP System Address Space JCL Procedure

```
//T18OTCP PROC MODULE='TCPIP',PARMS='NOSPIE/',PROFILE=PROFILE
//*
//* TCP/IP System Address Space for the OpenEdition MVS environment
//*
//TCPIP EXEC PGM=MVPMAIN,
//             PARM='&MODULE,ERRFILE(SYSERR),HEAP(512),&PARMS',
//             REGION=0M,TIME=1440
//STEPLIB   DD DSN=TCPIP.ITSC.LINKLIB,DISP=SHR
//          DD DSN=TCPIP.SEZATCP,DISP=SHR
```

© Copyright IBM Corp. 1996

```
//SYSPRINT  DD SYSOUT=*,DCB=(RECFM=VB,LRECL=137,BLKSIZE=0)
//SYSERR    DD SYSOUT=*,DCB=(RECFM=VB,LRECL=137,BLKSIZE=0)
//SYSERROR  DD SYSOUT=*,DCB=(RECFM=VB,LRECL=137,BLKSIZE=0)
//SYSDEBUG  DD SYSOUT=*,DCB=(RECFM=VB,LRECL=240,BLKSIZE=0)
//PROFILE   DD DSN=TCPIP.T18OE.TCPPARMS(&PROFILE.),DISP=SHR
//SYSTCPD   DD DSN=TCPIP.T18OE.TCPPARMS(TCPDATA),DISP=SHR
```

E.1.2 T18OROUT - OE Stack RouteD Server JCL Procedure

```
//T18OROUT PROC MODULE=ROUTED,PARMS='/-t',
//             GWDS=GATEWAYS
//*
//* RouteD server on the OpenEdition stack T18OTCP
//*
//ROUTED   EXEC PGM=&MODULE,
//             PARM='&PARMS',
//             REGION=4096K,TIME=1440
//STEPLIB   DD DSN=TCPIP.ITSC.LINKLIB,DISP=SHR
//          DD DSN=TCPIP.SEZATCP,DISP=SHR
//SYSPRINT  DD SYSOUT=*
//SYSERR    DD SYSOUT=*
//GATEWAYS  DD DSN=TCPIP.T18OE.TCPPARMS(&GWDS.),DISP=SHR
//SYSTCPD   DD DSN=TCPIP.T18OE.TCPPARMS(TCPDATA),DISP=SHR
//SERVICES  DD DSN=TCPIP.T18OE.ETC.SERVICES,DISP=SHR
//MSROUTED  DD DSN=TCPIP.SEZAINST(EZARTMSG),DISP=SHR,FREE=CLOSE
```

E.1.3 T18ODNS - OE Stack Caching-Only Name Server JCL Procedure

```
//T18ODNS PROC MODULE='NSMAIN',PARMS='TRACE'
//*
//* Turn on SMSG support
//*
//SETSMSG EXEC PGM=SETSMSG,PARM=ON
//SYSPRINT DD SYSOUT=*
//OUTPUT   DD SYSOUT=*
//SYSIN    DD DUMMY
//*
//* Start the caching-only name server
//*
//NAMESRV EXEC PGM=MVPMAIN,
//             PARM='&MODULE,&PARMS',
//             REGION=4096K,TIME=1440
//STEPLIB  DD DSN=TCPIP.SEZATCP,DISP=SHR
//SYSDNSD  DD DSN=TCPIP.T18OE.TCPPARMS(NSMAIN),DISP=SHR
//SYSDNSC  DD DSN=TCPIP.T18OE.TCPPARMS(NSCACHE),DISP=SHR
//MSNSMAIN DD DISP=SHR,DSN=TCPIP.SEZAINST(MSNSMAIN)
//SYSIN    DD DUMMY
//SYSPRINT DD SYSOUT=*
//SYSERR   DD SYSOUT=*
//SYSDEBUG DD SYSOUT=*
//LOG      DD SYSOUT=*
//SYSTCPD  DD DISP=SHR,DSN=TCPIP.T18OE.TCPPARMS(TCPDATA)
```

E.1.4 TCPDATA - OE Stack TCPIP.DATA

```
; ********************************************************************
; *
; *   OpenEdition MVS stack on MVS18 - T18OTCP
; *
; *   TCPIP.T18OE.TCPPARMS(TCPDATA)
; *
; *   This TCPIP.DATA file is used by non-OE applications that
; *   run on the T18OTCP stack:
; *
; *   - T18OROUT RouteD server
; *   - T18ODNS  Caching-only name server
; *   - TSO user that uses client commands like NETSTAT or OBEYFILE
; *
; *   The file is explicitly allocated via a SYSTCPD DD statement.
; *
; ********************************************************************
;
Datasetprefix TCPIP.T18OE       ;
TcpipJobname T18OTCP            ; For non-OE apps (RouteD and DNS)
Messagecase mixed               ;
HostName mvs18oe                ; Same name as in /etc/resolv.conf
DomainOrigin itso.ral.ibm.com   ;
NSinterAddr  14.0.0.0           ; Use our own caching-only DNS first
NSinterAddr  9.24.104.108       ; Then go to rsserver.itso.ral.ibm.com
NSportAddr 53                   ;
ResolveVia UDP                  ;
ResolverTimeout 10              ; Don't wait too long
ResolverUdpRetries 1            ;
Loaddbcstables sjiskanji        ; Load Sjiskanji DBCS tables
```

E.1.5 PROFILE - OE Stack PROFILE.TCPIP

```
; ********************************************************************
; *
; *   OpenEdition MVS stack on MVS18 - T18OTCP
; *   ----------------------------------------
; *
; *   This file is TCPIP.T18OE.TCPPARMS(PROFILE)
; *   JCL procedure for this stack is T18OETCP in TCPIP.PROCLIB
; *
; *   DATASETPREFIX is TCPIP.T18OE
; *
; *   PARMLIB is TCPIP.T18OE.TCPPARMS
; *
; ********************************************************************
;
; ********************************************************************
; * Buffer pool definitions.  Only TINYDATABUFFERS are specified,
; * the default value seems to be too low.
; ********************************************************************
;
TINYDATABUFFERPOOLSIZE      500
;
; ********************************************************************
; * ASSORTED Parameters
; ********************************************************************
;
ASSORTEDPARMS
```

```
  IGNOREREDIRECT    ; Ignore ICMP redirect packets
  MESSAGECASE MIXED ; TCP/IP Messages will be displayed in mixed case.
  TCPIPSTATISTICS   ; Print TCP/IP counters.
ENDASSORTEDPARMS
;
; ********************************************************************
; * KEEP alive options, defining the contents and timing of
; * KEEP alive messages sent over a TCP connection
; ********************************************************************
;
KEEPALIVEOPTIONS
 INTERVAL 120       ; Send message every 120 minutes
 SENDGARBAGE TRUE   ; Send random data and invalid sequence number
ENDKEEPALIVEOPTIONS
;
; ********************************************************************
; * Inform following users about serious run-time conditions
; ********************************************************************
;
INFORM
  ALFREDC
ENDINFORM
;
; ********************************************************************
; * Allow use of Obeyfile and Raw sockets
; ********************************************************************
;
OBEY
  T18OROUT                  ; ROUTED
  ALFREDC                   ;
  OMVS                      ; OMVS Raw sockets
ENDOBEY
;
; ********************************************************************
; * SMF record type 118 subtypes for this stack
; ********************************************************************
;
  SMFPARMS 121 122 123
;  121            ; sub type field in API initialization
;  122            ; sub type field in API termination
;  123            ; sub type field for FTP or TELNET client
;  LAODEXIT       ; Call TCPCNSMF
;
;
; ********************************************************************
; * Problems? call your system contact: SNMP object SYSCONTACT
; ********************************************************************
;
SYSCONTACT
  Alfred B Christensen 301-4564
ENDSYSCONTACT
;
; ********************************************************************
; * Know where your system is running: SNMP object SYSLOCATION
; ********************************************************************
;
SYSLOCATION
  MVS18 OE-Stack ITSO RALEIGH, B678, 1001 Winstead Drive, Cary
ENDSYSLOCATION
```

```
;
; ********************************************************************
; * Dataset prefix
; ********************************************************************
;
DATASETPREFIX TCPIP.T18OE
;
; ********************************************************************
; * Virtual Device and link definitions
; ********************************************************************
;
DEVICE DEVVIPA1  VIRTUAL 0
LINK   LINKVIPA1 VIRTUAL 0 DEVVIPA1
;
; ********************************************************************
; * IUCV definition TO MVS18 stack A (T18ATCP)
; ********************************************************************
;
DEVICE DEVTAA  IUCV XYZZY XYZZY T18ATCP B
LINK   LINKTAA IUCV 0   DEVTAA
;
; ********************************************************************
; * Tokenring interface on 3172-3 running ICP
; ********************************************************************
;
DEVICE DEVTR1  LCS           3OE   NETMAN
LINK   TR1     IBMTR         1     DEVTR1
;
; ********************************************************************
; * Start these non-OE servers during initialization
; * These procedures can be found in TCPIP.PROCLIB
; ********************************************************************
;
AUTOLOG
   T18OROUT                      ; ROUTED server
   T18ODNS                       ; Caching-only name server
ENDAUTOLOG
;
; ********************************************************************
; * THE PORTNUMBER assignment for the servers
; * and the PROTOCOL used on the port.
; * OMVS means the server is running in OpenEdition MVS
; ********************************************************************
;
PORT
    7 UDP  OMVS                  ; OE Echo
    7 TCP  OMVS                  ; OE Echo
    9 UDP  OMVS                  ; OE Discard
    9 TCP  OMVS                  ; OE Discard
   13 UDP  OMVS                  ; OE Daytime
   13 TCP  OMVS                  ; OE Daytime
   19 UDP  OMVS                  ; OE Chargen
   19 TCP  OMVS                  ; OE Chargen
   20 TCP  OMVS                  ; OE FTP Server data port
   21 TCP  OMVS                  ; OE FTP Server control port
 2020 TCP  OMVS                  ; OE FTP Alternate Server data port
 2021 TCP  OMVS                  ; OE FTP Alternate Server control port
   23 TCP  OMVS                  ; OE Telnet server
   37 TCP  OMVS                  ; OE Timeserver
```

```
   37 UDP  OMVS                  ; OE Timeserver
   53 TCP  T18ODNS               ; Caching-only name server
   53 UDP  T18ODNS               ; Caching-only name server
   80 TCP  OMVS                  ; OE WEB server
   111 UDP OMVS                  ; OE Portmapper Server
   111 TCP OMVS                  ; OE Portmapper Server
   123 TCP OMVS                  ; OE Web server ???
   512 TCP OMVS                  ; OE Remote Execution Server
   513 TCP OMVS                  ; OE Rlogin Server
   514 TCP OMVS                  ; OE Remote Shell Server
   514 UDP OMVS                  ; OE SyslogD Server
   520 UDP T18OROUT              ; RouteD Server
  8000 TCP OMVS                  ; OE WEB server
;
; ********************************************************************
; * THE PORTRANGE statement defines the ephemeral port number
; * range that is reserved for use by OMVS
; ********************************************************************

PORTRANGE 10000 2000 TCP OMVS    ; TCP
PORTRANGE 10000 2000 UDP OMVS    ; UDP

; ********************************************************************
; * HOME IP addresses for the LINKS of this stack
; ********************************************************************
;
HOME
   192.168.253.1   LINKVIPA1    ; VIPA link 1
   9.24.104.126    TR1          ; Tokenring interface
   192.168.254.1   LINKTAA      ; IUCV Link to MVS18 A T18ATCP
;
; ********************************************************************
; * RouteD Routing statements for dynamic routing
; * Static definitions are in TCPIP.T18OE.TCPPARMS(GATEWAYS)
; ********************************************************************
;
BSDROUTINGPARMS false
;
;    LINK     maxmtu   METRIC   SUBNET MASK     DESTINATION ADDRESS
;                                                (point to point links)
;
     LINKVIPA1 2000       0     255.255.255.0   0
     TR1       2000       0     255.255.255.0   0
     LINKTAA   2000       0     255.255.255.0   192.168.254.2
ENDBSDROUTINGPARMS
;
; ********************************************************************
; * We do not specify an INTERNALCLIENTPARMS section or
; * a VTAM section, because
; * this stack does not support tn3270 clients.  Port 23 is
; * reserved to OMVS for the OE telnet server.
; ********************************************************************
;
; ********************************************************************
; * Start devices when initialized
; ********************************************************************
;
 START DEVTR1               ; TR interface
 START DEVTAA               ; IUCV link to MVS18 A T18ATCP
```

E.1.6 NSMAIN - OE Stack Name Server NSMAIN.DATA

```
; ********************************************************************
; *
; * Caching-only name server on the OpenEdition stack on T18OTCP
; *
; * File name is TCPIP.T18OE.TCPPARMS(NSMAIN)
; *
; * Name server is started via procedure T18ODNS in TCPIP.PROCLIB
; *
; ********************************************************************
;
CACHINGONLY SYSDNSC           ; Use DD-name SYSDNSC as NSCACHE
NEGATIVECACHING               ; Cache will store negative query answers
STANDARDQUERYCACHE     100    ; The number of Standard queries to cache
INTERMEDIARYQUERYCACHE  50    ; The number of Intermediary queries to cache
INVERSEQUERYCACHE        5    ; The number of Inverse queries to cache
LRUTIME                300    ; Time an entry must be in the cache before it
HOSTNAMECASE         LOWER    ; Host names in DB2 are in lowercase
DOMAINNAMEPORT          53    ; Listen on port 53 (default)
UDPONLY                       ; Use UDP only when contacting a remote NS
UDPRETRYINTERVAL         5    ; After 5 seconds, try next remote NS
TRACE         NOTICE QUEUE    ; Display query origination and query name
```

E.1.7 NSCACHE - OE Stack Name Server NSCACHE.DATA

```
.    9999999 IN NS rsserver.itso.ral.ibm.com
rsserver.itso.ral.ibm.com 9999999 IN A 9.24.104.108
```

E.1.8 GATEWAYS - OE Stack RouteD Gateways Data Set

```
; ********************************************************************
; *
; * Gateway definitions for the OpenEdition stack on MVS18
; *
; * File name is TCPIP.T18OE.TCPPARMS(GATEWAYS)
; *
; ********************************************************************
;
; interval timers
;
options interface.scan.interval 30
options interface.poll.interval 15
;
net  0.0.0.0       gateway 9.24.104.1    metric 1 passive
```

E.1.9 REXX To Switch TSO User to Non-OE Stack

```
/* REXX */
/*********************************************************************/
/*                                                                   */
/* Switch TSO Address Space to use the non-OE stack                  */
/* on MVS18: T18ATCP.                                                */
/* Subsequent NETSTAT or OBEY commands will be directed towards      */
/* the T18ATCP stack.                                                */
/*                                                                   */
/* To switch to the OE stack (T18OTCP) - run REXX program T18O       */
/*                                                                   */
/*********************************************************************/
Say 'Switching to T18ATCP stack (The system default)'
```

```
msgstat = msg()
z = msg("OFF")
"FREE FI(SYSTCPD)"
"ALLOC FI(SYSTCPD) DA('SYS1.TCPPARMS(TCPDATA)') SHR"
z = msg(msgstat)

exit(0)
```

E.1.10 Rexx To Switch TSO User to OE Stack

```
/* REXX */
/*********************************************************************/
/*                                                                   */
/* Switch TSO Address Space to use the OpenEdition MVS stack         */
/* on MVS18: T18OTCP.                                                */
/* Subsequent NETSTAT or OBEY commands will be directed towards      */
/* the T18OTCP stack.                                                */
/*                                                                   */
/* To switch to the non-OE stack (T18ATCP) - run REXX program T18A   */
/*                                                                   */
/*********************************************************************/
Say 'Switching to T18OTCP stack'

msgstat = msg()
z = msg("OFF")
"FREE FI(SYSTCPD)"
"ALLOC FI(SYSTCPD) DA('TCPIP.T18OE.TCPPARMS(TCPDATA)') SHR"
z = msg(msgstat)

exit(0)
```

E.2 OpenEdition Configuration and Files

The files we include in this section are the full set of OE resolver configuration files and a number of the OpenEdition initialization files and shell scripts.

The OE resolver configuration files were:

```
/etc/resolv.conf
/etc/services
/etc/protocol
/etc/hosts
TCPIP.OMVS.STANDARD.TCPXLBIN
```

E.2.1 /etc/resolv.conf

```
;*********************************************************************
;                                                                    *
;     /etc/resolv.conf                                               *
;                                                                    *
;     OpenEdition MVS Resolver on mvs18                              *
;                                                                    *
;     This file is used by the OpenEdition MVS resolver code in      *
;     all OE socket applications.                                    *
;                                                                    *
;     There is no TCPIPJOBNAME in this file.  The AF_INET PFS        *
;     decides on its own if it should use a TCP/IP stack or an       *
;     AnyNet stack, and in the case of more TCP/IP stacks, which one.  *
```

```
;                                                                    *
;*********************************************************************
;
Datasetprefix TCPIP.OMVS       ; Only used for TCPXLBIN data set
Messagecase mixed              ;
HostName mvs18oe               ; MVS18 OE environment host name
DomainOrigin itso.ral.ibm.com  ;
NSinterAddr  9.24.104.126      ; Use our own caching-only DNS first
NSinterAddr  9.24.104.108      ; Then go to rsserver.itso.ral.ibm.com
NSportAddr 53                  ;
ResolveVia UDP                 ;
ResolverTimeout 10             ; Don't wait too long
ResolverUdpRetries 1           ;
```

E.2.2 /etc/services

```
#
# Network services, Internet style
#
echo            7/tcp
echo            7/udp
discard         9/tcp          sink null
discard         9/udp          sink null
systat          11/tcp         users
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
qotd            17/tcp         quote
chargen         19/tcp         ttytst source
chargen         19/udp         ttytst source
ftp             21/tcp
telnet          23/tcp
ttelnet       9023/tcp                        # Trace telnet
mytelnet      2023/tcp                        # Alfred test...
smtp            25/tcp         mail
time            37/tcp         timserver
time            37/udp         timserver
rlp             39/udp         resource        # resource location
nameserver      42/tcp         name            # IEN 116
whois           43/tcp         nicname
domain          53/tcp         nameserver      # name-domain server
domain          53/udp         nameserver
mtp             57/tcp                         # deprecated
tftp            69/udp
rje             77/tcp         netrjs
finger          79/tcp
www             80/tcp                         # web server
link            87/tcp         ttylink
supdup          95/tcp
hostnames       101/tcp        hostname        # usually from sri-nic
#csnet-cs       105/?
pop             109/tcp        postoffice
sunrpc          111/tcp        portmap oportmap
sunrpc          111/udp        portmap oportmap
auth            113/tcp        authentication
sftp            115/tcp
uucp-path       117/tcp
nntp            119/tcp        readnews untp   # USENET News Transfer Protocol
snmp            161/udp                        # snmp request port
```

```
snmp-trap       162/udp                         # snmp monitor trap port
#
# UNIX specific services
#
exec            512/tcp
texec          9512/tcp                         # Trace rexecd
biff            512/udp         comsat
login           513/tcp
tlogin         9513/tcp                         # Trace rlogind
who             513/udp         whod
shell           514/tcp         cmd             # no passwords used
tshell         9514/tcp                         # Trace rshd
syslog          514/udp
printer         515/tcp         spooler         # line printer spooler
talk            517/udp
ntalk           518/udp
efs             520/tcp                         # for LucasFilm
timed           525/udp         timeserver
tempo           526/tcp         newdate
courier         530/tcp         rpc
conference      531/tcp         chat
netnews         532/tcp         readnews
netwall         533/udp                         # -for emergency broadcasts
uucp            540/tcp         uucpd           # uucp daemon
remotefs        556/tcp         rfs_server rfs  # Brunhoff remote filesystem

ingreslock      1524/tcp
#
# Start of IBM added services ...
#
route           520/udp         router routed
ncprout         580/udp         ncproute

#
# RVD service
#
rvd-control     531/udp                         # rvd control port
#
# Andrew File System services
#
filesrv         2001/tcp
console         2018/udp
venus.itc       2106/tcp

# For file server backup and migration
client          2030/tcp

#
# Andrew File System Authenticated services
#
vexec           712/tcp         vice-exec
vlogin          713/tcp         vice-login
vshell          714/tcp         vice-shell

# For the Venus process.
venus.itc       2106/tcp
rauth2          2001/udp
rfilebulk               2002/udp
rfilesrv                2003/udp
```

```
ropcons         2115/udp
# The following are assigned in pairs and the bulk must be the srv +1
rupdsrv         2131/udp
rupdbulk                2132/udp
rupdsrv1                2133/udp
rupdbulk1               2134/udp

#
#     Kerberos services
#

klogin          543/tcp           # Kerberos aythenticated rlogin
kerberos        750/udp      kdc  # Kerberos aythentication--udp
kerberos        750/tcp      kdc  # Kerberos aythentication--tcp
kerboros_master 751/udp           # Kerberos aythentication
kerberos_master 751/tcp           # Kerberos aythentication
passwd_server   752/udp           # Kerberos passwd server
userreg_server  753/tcp           # Kerberos userreg serrver
kpop            1109/tcp          # Pop with Kerberos
knetd         2053/tcp                # Kerberos de-multiplexor
kshell          544/tcp         cmd     # and remote shell
eklogin         2105/tcp                # Kerberos encrypted rlogin
krb_prop        754/tcp                 # Kerboros slpave propagation
erlogin         888/tcp                 # Login and environment passing
#
#
#       Kerberos sample server
#
sample        906/tcp                 # Kerberos sample app server
sample                          906/udp                #for kerberos simple test
```

E.2.3 /etc/protocol

```
# @(#)protocols 7.2 88/06/28 12:40:03
#
# /etc/protocols file for AIX
#
# offical name, protocol number, aliases

ip              0               # dummy for IP
icmp            1               # control message protocol
# ggp           2               # gateway¬2 (deprecated)
tcp             6               # tcp
# egp           8               # exterior gateway protocol
# pup           12              # pup
udp             17              # user datagram protocol
# idp           22              # xns idp
```

E.2.4 /etc/hosts

```
#
# OE Resolver /etc/hosts file on mvs18oe.
#
# The format of this file is:
#
# Internet Address      Hostname   Aliases     # Comments
#
# Items are separated by any number of blanks and/or tabs.  A '#'
# indicates the beginning of a comment; characters up to the end of the
```

```
# line are not interpreted by routines which search this file.  Blank
# lines are allowed in this file.

9.24.104.126    mvs18oe mvsoe # OE host
192.168.210.1   mvs18an       # AnyNet MVS host
192.168.210.8   mypcaa        # AnyNet gw host
9.24.104.79     mypc          # A workstation
```

E.2.5 /etc/rc

```
# Initialization shell script, pathname = /etc/rc

# Initial setup for OpenEdition MVS
export _BPX_JOBNAME='ETCRC'

# Setup utmpx file
>/etc/utmpx
chmod 644 /etc/utmpx

# Reset all slave tty files
chmod 666 /dev/tty*
chown 0 /dev/tty*

# Setup write, talk, mesg utilities
chgrp TTY   /bin/write
chgrp TTY   /bin/mesg
chgrp TTY   /bin/talk
chmod 2755  /bin/write
chmod 2755  /bin/mesg
chmod 2755  /bin/talk

# Invoke vi recovery
mkdir -m 777 /etc/recover
/usr/lib/exrecover

echo /etc/rc script executed, date
```

E.2.6 /etc/init.options

```
-a  120                          timeout = 120 seconds
-t  1                            terminate shell = yes
-sc /etc/rc                      shell script = /etc/rc
-e  TZ=EST5EDT                   TZ environment variable
-sh /bin/sh                      shell = /bin/sh
```

E.2.7 /etc/profile

```
if [ -z "$STEPLIB" ] && tty -s;
then
    echo " - - - - - - - - - - - - - - - - - - - - - - - - - - - - "
    echo " - Improve performance by preventing the propagation - "
    echo " - of TSO/E or ISPF STEPLIBs                         - "
    echo " - - - - - - - - - - - - - - - - - - - - - - - - - - - - "
    export STEPLIB=none
    exec sh -L
fi
TZ=EST5EDT
PATH=/bin:.
NLSPATH=/usr/lib/nls/msg/%L/%N
LANG=C
```

```
MAIL=/usr/mail/$LOGNAME
export TZ PATH NLSPATH MAIL LANG
umask 022
readonly LOGNAME
#
# Version of programs and runtime library used by c89/cc/c++:
# ===========================================================
#
#   Compiler:                                 Version Release Mod:
#   ---------                                 --------------------
    export _C89_CVERSION="0x13010000"         # V3R1M0
    export  _CC_CVERSION="0x13010000"
#
#   Prelinker and runtime library:
#   ------------------------------
    export _C89_PVERSION="0x11050000"         # V1R5M0 (dynamic default)
    export  _CC_PVERSION="0x11050000"
    export _CXX_PVERSION="0x11050000"
#
# High-Level Qualifier "prefixes" for data sets used by c89/cc/c++:
# =================================================================
#
#   C/C++ Compiler:                           {_CVERSION} default...
#   ---------------                           ----------------------
    export _C89_CLIB_PREFIX="CBC.V3R1M0"      # 0x13010000
    export  _CC_CLIB_PREFIX="CBC.V3R1M0"
#
#   Prelinker and runtime library:            {_PVERSION} default...
    export _C89_PLIB_PREFIX="CEE.V1R5M0"      # 0x11050000
    export  _CC_PLIB_PREFIX="CEE.V1R5M0"
    export _CXX_PLIB_PREFIX="CEE.V1R5M0"
#
#   MVS system:
#   -----------
    export _C89_SLIB_PREFIX="SYS1"
    export  _CC_SLIB_PREFIX="SYS1"
    export _CXX_SLIB_PREFIX="SYS1"
#
# Esoteric unit for data sets:
# ============================
#
#   Unit for (unnamed) work data sets:
#   ----------------------------------
    export _C89_WORK_UNIT="SYSDA"
    export  _CC_WORK_UNIT="SYSDA"
    export _CXX_WORK_UNIT="SYSDA"
```

E.2.8 SYS1.PARMLIB(BPXPRMxx) Parmlib Member

```
MAXPROCSYS(200)
MAXPROCUSER(25)
MAXUIDS(200)
MAXFILEPROC(256)
MAXPTYS(256)
MAXTHREADTASKS(500)
MAXTHREADS(500)
CTRACE(CTIBPX00)
SUPERUSER(BPXROOT)
STEPLIBLIST('/system/steplib')
```

```
/* The HFS file system definitions and static mounts                */

FILESYSTYPE TYPE(HFS)
            ENTRYPOINT(GFUAINIT)
ROOT FILESYSTEM('SMS.OMVS52.SA18.ROOT')
      TYPE(HFS)
      MODE(RDWR)
MOUNT FILESYSTEM('SMS.IMW.SIMWHFS.SA18')
      MOUNTPOINT('/usr/lpp/internet')
      TYPE(HFS)
      MODE(RDWR)
MOUNT FILESYSTEM('SMS.TCPIP.XWIN.SA18')
      MOUNTPOINT('/usr/lpp/tcpip')
      TYPE(HFS)
      MODE(RDWR)
MOUNT FILESYSTEM('SMS.SIOE.LOCAL.SA18')
      MOUNTPOINT('/usr/lpp/dfs/local')
      TYPE(HFS)
      MODE(RDWR)
MOUNT FILESYSTEM('SMS.SIOE.GLOBAL.SA18')
      MOUNTPOINT('/usr/lpp/dfs/global')
      TYPE(HFS)
      MODE(RDWR)
MOUNT FILESYSTEM('SMS.DCEAS.SA18')
      MOUNTPOINT('/usr/lpp/dceas')
      TYPE(HFS)
      MODE(RDWR)
MOUNT FILESYSTEM('SMS.EUV.LOCAL.SA18')
      MOUNTPOINT('/usr/lpp/dce/local')
      TYPE(HFS)
      MODE(RDWR)
MOUNT FILESYSTEM('SMS.EUV.GLOBAL.SA18')
      MOUNTPOINT('/usr/lpp/dce/global')
      TYPE(HFS)
      MODE(RDWR)
MOUNT FILESYSTEM('SMS.OMVS.USERS')
      TYPE(HFS)
      MODE(RDWR)
      MOUNTPOINT('/u')
MOUNT FILESYSTEM('SMS.OMVS.ALFREDC')
      TYPE(HFS)
      MODE(RDWR)
      MOUNTPOINT('/u/alfredc')
MOUNT FILESYSTEM('SMS.OMVS.TCPIPWEB.HFS')
      TYPE(HFS)
      MODE(RDWR)
      MOUNTPOINT('/u/tcpipweb')
MOUNT FILESYSTEM('SMS.OMVS.CS2201.HFS')
      TYPE(HFS)
      MODE(RDWR)
      MOUNTPOINT('/u/cs2201')
MOUNT FILESYSTEM('SMS.OMVS.LOSERT')
      TYPE(HFS)
      MODE(RDWR)
      MOUNTPOINT('/u/losert')
MOUNT FILESYSTEM('SMS.OMVS.SUZUKI')
      TYPE(HFS)
      MODE(RDWR)
```

```
      MOUNTPOINT('/u/suzuki')
MOUNT FILESYSTEM('SMS.OMVS.REIKO')
      TYPE(HFS)
      MODE(RDWR)
      MOUNTPOINT('/u/reiko')
MOUNT FILESYSTEM('SMS.OMVS.HDM')
      TYPE(HFS)
      MODE(RDWR)
      MOUNTPOINT('/u/hdm')
MOUNT FILESYSTEM('SMS.OMVS.JURI')
      TYPE(HFS)
      MODE(RDWR)
      MOUNTPOINT('/u/juri')
MOUNT FILESYSTEM('SMS.OMVS.KITTY')
      TYPE(HFS)
      MODE(RDWR)
      MOUNTPOINT('/u/kitty')
MOUNT FILESYSTEM('SMS.OMVS.SCADDEN')
      TYPE(HFS)
      MODE(RDWR)
      MOUNTPOINT('/u/scadden')

/* AF_UNIX file system for local sockets                              */

FILESYSTYPE TYPE(UDS)
            ENTRYPOINT(BPXTUINT)
NETWORK DOMAINNAME(AF_UNIX)
        DOMAINNUMBER(1)
        MAXSOCKETS(2000)
        TYPE(UDS)

/* AF_INET file system for TCP/IP sockets                             */
/* We use the converged socket support - BPXTCINT                     */

FILESYSTYPE TYPE(CINET)
            ENTRYPOINT(BPXTCINT)
NETWORK DOMAINNAME(AF_INET)
        DOMAINNUMBER(2)
        MAXSOCKETS(10000)
        TYPE(CINET)
        INADDRANYPORT(10000)
        INADDRANYCOUNT(2000)
SUBFILESYSTYPE NAME(T180TCP)
               TYPE(CINET)
               ENTRYPOINT(BPXTIINT)
               DEFAULT
SUBFILESYSTYPE NAME(RAISOCK)
               TYPE(CINET)
               ENTRYPOINT(ISTOEPIT)
SUBFILESYSTYPE NAME(LINET)
               TYPE(CINET)
               ENTRYPOINT(BPXTLINT)
```

E.3 AnyNet MVS Configuration Data Sets

For AnyNet MVS, we used the following configuration data sets:

```
TCPIP.ANYNET.ETC.PROTO
TCPIP.ANYNET.ETC.SERVICES
TCPIP.ANYNET.HOSTS.ADDRINFO
TCPIP.ANYNET.HOSTS.SITEINFO
TCPIP.ANYNET.RESOLVER
TCPIP.ANYNET.STANDARD.TCPXLBIN
```

ETC.PROTO and ETC.SERVICES were standard setup based on the samples from SYS1.SAMPLIB. HOSTS.ADDRINFO and HOSTS.SITEINFO were maintained via the same method as we used for the TCP/IP for MVS counterparts.

E.3.1 RESOLVER

```
;*********************************************************************
;                                                                    *
;     OpenEdition MVS stack on MVS18 - ANYNET                        *
;                                                                    *
;*********************************************************************
;
DATASETPREFIX TCPIP.ANYNET
TCPIPJOBNAME RAISOCK
MESSAGECASE MIXED
HostName MVS18AN
DomainOrigin itso.ral.ibm.com
NSinterAddr  9.24.104.108
NSportAddr 53
ResolveVia UDP
ResolverTimeout 20
ResolverUdpRetries 1
```

E.3.2 ENVVAR Data Set

```
ADDRINFO='TCPIP.ANYNET.HOSTS.ADDRINFO'
SITEINFO='TCPIP.ANYNET.HOSTS.SITEINFO'
DNS_XLATE_TABLE='TCPIP.ANYNET.STANDARD.TCPXLBIN'
HOSTS_FILE_FORMAT=MVSTCP
HOSTNAME=MVS18AN
ETC_PROTOCOLS='TCPIP.ANYNET.ETC.PROTO'
ETC_RESOLV='TCPIP.ANYNET.RESOLVER'
ETC_SERVICES='TCPIP.ANYNET.ETC.SERVICES'
GROUP_NAME=SNACKETS
SXMODE_DEFAULT=SNACKETS
OPEN_EDITION=YES
OE_INADDRANY_COUNT=2000
OE_INADDRANY_PORT=10000
```

Appendix F. Special Notices

This publication is intended to help customers to plan for, install, customize, and operate TCP/IP for MVS including the TCP/IP for MVS OpenEdition Applications Feature, AnyNet MVS and the Internet Connection Server for MVS in an OpenEdition environment. The information in this publication is not intended as the specification of any programming interfaces that are provided by OpenEdition, TCP/IP for MVS, AnyNet MVS, or the Internet Connection Server for MVS. See the PUBLICATIONS section of the IBM Programming Announcement for OpenEdition, TCP/IP for MVS, AnyNet MVS and the Internet Connection Server for MVS for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM (″vendor″) products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AD/Cycle
AIX
AIX/6000
AnyNet
APPN
AT
C/MVS
C/370
CICS
DatagLANce

© Copyright IBM Corp. 1996

DB2
ES/9000
IBM
Language Environment
MVS/ESA
OpenEdition
OS/390
PS/2
S/390
VTAM
DFSMS/MVS
FFST
IMS
MVS
NetView
OS/2
OS/400
RACF
SAA

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Microsoft, Windows, and the Windows 95 logo
are trademarks or registered trademarks of Microsoft Corporation.

Other trademarks are trademarks of their respective companies.

Appendix G. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

G.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How To Get ITSO Redbooks” on page 249.

- *TCP/IP V3R1 for MVS Implementation Guide*, GG24-3687
- *A Beginner's Guide to MVS TCP/IP Socket Programming*, GG24-2561
- *OpenEdition MVS Installation and Customization Starter Kit*, SG24-4529
- *RACF Support for Open Systems Technical Presentation Guide*, GG26-2005-00
- *Multiprotocol Transport Networking (MPTN) Architecture Tutorial and Product Implementations*, SG24-4170-01
- *AnyNet: SNA over TCP/IP Installation and Interoperability*, GG24-4395-00
- *AnyNet: Sockets over SNA NetBIOS over SNA Installation and Interoperability*, GG24-4396-00

A complete list of International Technical Support Organization publications, known as redbooks, with a brief description of each, may be found in:

International Technical Support Organization Bibliography of Redbooks, GG24-3070.

G.2 Other Publications

These publications are also relevant as further information sources:

- 5655-HAL IBM TCP/IP V3 MVS
  - *TCP/IP V3R1 for MVS: Diagnosis Guide*, LY43-0105-01 (available to IBM-licensed customers only)
  - *TCP/IP for MVS V3R1 Messages and Codes*, SC31-7132-02
  - *TCP/IP for MVS V3R1 Customization Guide*, SC31-7134-02
  - *TCP/IP for MVS V3R1 Programmer's Guide*, SC31-7135-01
  - *TCP/IP for MVS V3R1 User's Guide*, SC31-7136-01
  - *TCP/IP for MVS V3R1 Application Programming Guide*, SC31-7187-00
  - *TCP/IP for MVS V3R1 Performance and Tuning*, SC31-7188-01
  - *TCP/IP for MVS V3R1 Planning*, SC31-7189-01
  - *TCP/IP V3R1 for MVS: Quick Reference*, SX75-0095-00
  - *TCP/IP V.3 for OE MVS Application Feature Guide*, SC31-8069-00
- 5655-068 MVS/ESA SP - JES2 V5
  - *MVS/ESA SP V5 APPC/MVS Handbook*, GC28-1504-00
  - *OpenEdition MVS POSIX.1 Conformance*, GC23-3011-02

© Copyright IBM Corp. 1996

  - *OpenEdition MVS POSIX.2 Conformance*, GC23-3012-01
  - *OpenEdition MVS Advanced Applications*, SC23-3017-02
  - *Assembler Callable Services for MVS/ESA*, SC23-3020-02
  - *Using REXX to Access OpenEdition MVS*, SC23-3803-01
  - *IBM COBOL for MVS & VM Compiler & Library*, GC26-4764-03
  - *Language Environment Rel 5 Fact Sheet*, GC26-4785-04
  - *LE Concepts Guide R4*, GC26-4786-03
  - *C/C++ for MVS V3R2.0 Compiler*, SC09-2002-01
  - *OpenEdition MVS C/C++ MVS/ESA Library*, SC23-3881-01
  - *Language Environment V1R5 Programming Reference*, SC26-3312-02
  - *Language Environment V1R5 Programming Guide*, SC26-4818-05
  - *Language Environment V1R5 Run-Time Migration Guide*, SC26-8232-01
  - *Language EnvironmentIR V1R5 Writing ILC Applications*, SC26-8351-01
  - *Understanding DCE Concepts*, GC09-1478-01
  - *OpenEdition DCE (MVS/ESA SP V5.2.2)*, SC09-1488-00
  - *OpenEdition MVS Messages and Codes*, SC23-3780-02
  - *OpenEdition MVS File System Reference*, SC23-3802-01
  - *OpenEdition MVS XPG4 Conformance Documentation*, GC23-3873-00
  - *OpenEdition MVS Communications*, SC23-3883-00
  - *C/MVS Library Reference: OpenEdition MVS*,SC23-3876-00
- 5655-069 MVS/ESA SP - JES3 V5
  - *MVS/ESA Introducing DCE*, GC09-1482-00
  - *Introducing OpenEdition MVS*, GC23-3010-02
  - *MVS/ESA SP Planning - Operation*, GC28-1441-01
  - *MVS/ESA SP V5 Planning - Operation*, GC28-1441-02
  - *MVS/ESA SP V5 JCL User's Guide*, GC28-1473-02
  - *MVS/ESA SP V5 JCL Reference*, GC28-1479-02
  - *MVS/ESA SP V5 System Messages Vol1*, GC28-1480-02
  - *MVS/ESA SP V5 System Messages Vol2*, GC28-1481-02
  - *MVS/ESA SP V5 System Messages Vol3*, GC28-1482-03
  - *MVS/ESA SP V5 System Messages Vol4*, GC28-1483-02
  - *MVS/ESA SP V5 System Messages Vol5*, GC28-1484-02
  - *MVS/ESA SP V5 System Codes*, GC28-1486-02
  - *MVS/ESA SP V5 JES3 Messages*, GC28-1489-01
  - *MVS/ESA SP Planning - APPC Management*, GC28-1503-01
  - *OpenEdition MVS User's Guide*, SC23-3013-02
  - *OpenEdition MVS Command Reference*, SC23-3014-02
  - *Planning: OpenEdition MVS*, SC23-3015-02

- *MVS/ESA SP V5 APPC/MVS Handbook*, GC28-1504-00
- *OpenEdition MVS POSIX.1 Conformance*, GC23-3011-02
- *OpenEdition MVS POSIX.2 Conformance*, GC23-3012-01
- *OpenEdition MVS Advanced Applications*, SC23-3017-02
- *ASSEMBLER Callable Services for MVS*, SC23-3020-02
- *Using REXX to Access OpenEdition MVS*, SC23-3803-01
- *MVS/ESA SP Diagnosis - Procedures*, LY28-1844-01 (available to IBM-licensed customers only)
- *IBM COBOL for MVS & VM Compiler & Library*, GC26-4764-03
- *Language Environment REL 5 Fact Sheet*, GC26-4785-04
- *LE Concepts Guide R4*, GC26-4786-03
- *C/C++ for MVS V3R2.0 Compiler*, SC09-2002-01
- *OpenEdition MVS C/C++ MVS/ESA Library*, SC23-3881-01
- *Language Environment V1R5 Programming Reference*, SC26-3312-02
- *Language Environment V1R5 Programming Guide*, SC26-4818-05
- *LAN Environment V1R5 Run-Time Migration Guide*, SC26-8232-01
- *Language Environment V1R5 Writing ILC Applications*, SC26-8351-01
- *Understanding DCE Concepts*, GC09-1478-01
- *OpenEdition DCE (MVS/ESA SP V5.2.2)*, SC09-1488-00
- *OpenEdition MVS Messages and Codes*, SC23-3780-02
- *RACF Callable Services V2R2*, GC23-3737-01
- *OpenEdition MVS File System Reference*, SC23-3802-01
- *OPEN EDITION MVS XPG4 Conformance Documentation*, GC23-3873-00
- *OpenEdition MVS Communications*, SC23-3883-00
- *C/MVS Library Reference: OpenEdition MVS*, SC23-3876-00

• 5695-117 ACF/VTAM V4 MVS/ESA
- *VTAM for MVS/ESA V4R3 Migration Guide*, GC31-6547-00
- *VTAM MVS LPS*, GC31-6553-00
- *VTAM for MVS/ESA V4R3 Release Guide*, GC31-6555-00
- *VTAM for MVS Glossary*, GC31-6556-00
- *APPC Application Suite V1 User's Guide*, SC31-6532-00
- *APPC Application Suite V1 Administration Guide*, SC31-6533-00
- *VTAM for MVS/ESA V4R3*, SC31-6546-00
- *VTAM for MVS Network Implementation*, SC31-6548-00
- *VTAM for MVS Operation Guide*, SC31-6549-00
- *VTAM for MVS Resource Definition Reference*, SC31-6552-00
- *Planning for Integrated Network*, SC31-8062-00
- *Estimating Storage for VTAM V4*, SK2T-6400-00
- *VTAM for MVS Operations Quick Reference*, SX75-0207-00

  - *VTAM for MVS/ESA V4R3 Customization Guide*, LY43-0068-00 (available to IBM-licensed customers only)
  - *VTAM for MVS/ESA V4R3 Data Areas*, LY43-0071-00 (available to IBM-licensed customers only)
  - *VTAM for MVS/ESA V4R3 Diagnosis Guide*, LY43-0069-00 (available to IBM-licensed customers only)
  - *VTAM for MVS/ESA V4R3 CMIP*, SC31-6544-00
  - *VTAM for MVS Resource Definition Samples*, SC31-6554-00
  - *APPC Application Suite V1 Programming*, SC31-6534-00
  - *VTAM for MVS/ESA V4R3 Programming*, SC31-6550-00
  - *VTAM for MVS/ESA V4R3 Programming*, SC31-6551-00
  - *VTAM AnyNet Feature Guide To Sockets, SNA*, SC31-6559-00
  - *VTAM AnyNet Feature Guide To SNA, TCP/IP*, SC31-6560-00
- 5655-156 Internet Connection Server MVS/ESA
  - *IBM Internet Connection Server Guide*, SC31-8204-00

How To Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at URL http://www.redbooks.ibm.com.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type GOPHER.WTSCPOK.ITSO.IBM.COM
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get lists of redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Home Page on the World Wide Web**

  http://w3.itso.ibm.com/redbooks

- **IBM Direct Publications Catalog on the World Wide Web**

  http://www.elink.ibmlink.ibm.com/pbl/pbl

  IBM employees may obtain LIST3820s of redbooks from this page.

- **ITSO4USA category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

© Copyright IBM Corp. 1996

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** (Do not send credit card information over the Internet) — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	bookshop at dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29554	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada (toll free)	1-800-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1) 415 855 43 29 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** - send note to softwareshop@vnet.ibm.com
- **On the World Wide Web**

Redbooks Home Page	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to announce@webster.ibmlink.ibm.com with the keyword subscribe in the body of the note (leave the subject line blank).

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

- **Please put me on the mailing list for updated versions of the IBM Redbook Catalog.**

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

- Invoice to customer number _____
- Credit card number _____

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

DO NOT SEND CREDIT CARD INFORMATION OVER THE INTERNET.

List of Abbreviations

AF	Addressing Family
AF_INET	Addressing Family, Internet
AF_UNIX	Addressing Family, UNIX
API	Application programming interface
DCE	Distributed Computing Environment
CINET	Common Inet
FTP	File Transfer Protocol
GID	UNIX Group ID
HFS	Hierarchical File System
HLQ	High Level Qualifier
IBM	International Business Machines Corporation
IP	Internet Protocol
ITSO	International Technical Support Organization
LFS	Logical File System
ONC	Open Network Computing
OCS	Outboard Communication Server
PADS	RACF Program Access to Data Sets
PFS	Physical File System
POSIX	Portable Operating System Interface - UNIX
REXEC	Remote Execution Protocol
RFC	Request For Comments
RPC	Remote Procedure Call
RSH	Remote Shell Protocol
RTL	Runtime Library
SD	Socket Descriptor
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol / Internet Protocol
UDP	User Datagram Protocol
UID	UNIX User ID
URL	Uniform Resource Locator
XPG	X/Open Portability Guide

© Copyright IBM Corp. 1996

Index

Special Characters

A

B

© Copyright IBM Corp. 1996

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Printed in U.S.A.